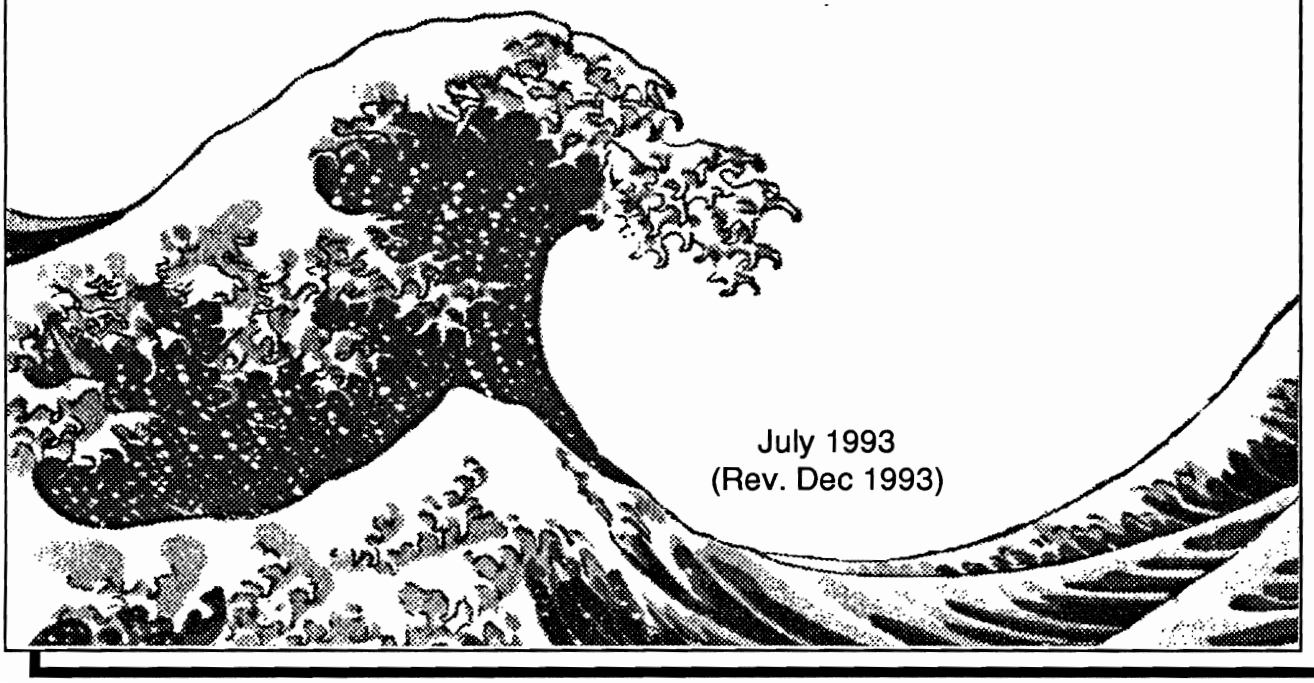
MOSS LANDING MARINE LABORATORIES

P.O. Box 450 Moss Landing, CA USA 95039-0450 (408) 633-3304

Introduction to MLML_DBASE Programs

William W. Broenkow and Richard E.Reaves

Moss Landing Marine Laboratories Technical Publication 93-1



July 1993
(Rev. Dec 1993)

Introduction to MLML_DBASE Programs

William W. Broenkow and Richard E. Reaves

Moss Landing Marine Laboratories

Moss Landing Marine Laboratories Technical Publication 93-1
Moss Landing, CA 95039

July 1993
(Rev Dec 1993)

Introduction to MLML_DBASE Programs

Table of Contents

List of Figures

List of Tables

Introduction to MLML_DBASE Programs

William W. Broenkow and Richard E. Reaves
Moss Landing Marine Laboratories

Abstract

The MLML_DBASE family of programs described here provides many of the algorithms used in oceanographic data reduction, general data manipulation and line graphs. These programs provide a consistent file structure for serial data typically encountered in oceanography. This introduction should provide enough general knowledge to explain the scope of the program and to run the basic MLML_DBASE programs. It is not intended as a programmer's guide.

Background

The MLML_DBASE programs are a group of FORTRAN and C programs written for VAX/VMS computers that work with a 2-dimensional (column and row) numerical database. These programs include routines for general statistics, time series analysis, data transformations, oceanographic functions, navigation, mapping, solar ephemeris, tidal prediction, and line and scatter plots. Both the database and the programs are extensions of the MLML "PLOT" programs written for HP Series 200 desktop computers. These programs operate on a single data structure. All data obtained by the MLML physical oceanography group since 1974 are stored in this format. This includes: CTD and hydrographic cast data, current meter records, meteorological time series from the Monterey Bay Aquarium and MLML, the Monterey Bay CALCOFI hydrographic data set, and the NOAA/NESDIS CZCS spectroradiometer and profiling data, maps and charts, and the GEOSECS and TTO oceanographic data.

All programs are available to all VAX users at Moss Landing Marine Laboratories, and we encourage users to develop their own programs to fulfill their specific needs. To do that requires a knowledge of FORTRAN, the MLML_DBASE data structure, and the subroutine library described later. I apparent to the reader that most of the MLML_DBASE programs are for data analysis rather than for plotting. However, because of historical tradition, this report will hereafter refer to MLML_DBASE as *PLOT*. The subprograms, whose formal names, for example, are MLML_FILE, or MLML_TRANSFORM, will be shortened to *FILE* and *TRANSFORM*.

Users of existing *PLOT* programs need to know little more about the *PLOT* data structure than that the data are essentially columns and rows of numbers. Each column of data is contained in a logical unit called a *Variable*. These variables hold the 1-dimensional numerical data along with descriptive information and parameters to scale plots. A collection of related *Variables* along with file *Header* information are organized into *Files*. These terms are explained in more detail later.

MLHPL

The MLML *PLOT* programs were originally written for Hewlett-Packard desktop computers in the interpretive HPL language which is not transportable to the VAX. To retain some of the HPL friendliness, which allowed command line calculations, we wrote the *MLHPL* interpreter, which uses HPL syntax to allow complex algebraic statements as inputs to *PLOT* programs. The main use of the *MLHPL* interpreter is in the *TRANSFORM* program (described later), which allows quite powerful data manipulation capabilities to users of the database.

MLHPL gives access to a library of more than 80 functions (Table 3). All of the SCOR/UNESCO oceanographic functions are included in the subroutine library (Appendix 3), and many of these are implemented in *MLHPL*. These functions are used to transform data: for example to calculate the time of sunrise from *Variables* holding latitude, longitude and date; to calculate salinity from conductivity, temperature and pressure; or more simply to multiply one *Variable* times another. The MLHPL interpreter is used in the *PLOT* program, *TRANSFORM*, as well as in a similar free-standing command line calculator program. *MLHPL* offers use of simple variables named A..Z. These variables may be used in *TRANSFORM*, and may be useful as a scratchpad to retain values for a variety of purposes.

Starting and Quitting PLOT

The program is invoked by the VMS symbol,

```
$ PLOT
```

or by typing the equivalent command

```
$ RUN MLML_DBASE$DIR:MLML_MAIN
```

When run this way, the user is prompted for the number of *Variables* (or columns) and the number of data *Elements* (or rows):

```
ENTER Number of Variables                    50
ENTER Number of Elements (max = 10000)     1000
```

This allocates (or dimensions) *Workspace* for 50,000 data elements. The user must have some idea of the best way to allocate memory to process a particular data set. If a file cannot be loaded because it is larger than the dimensioned size, the program must be restarted. It is useful to allocate more *Variables* than necessary for your largest file size to allow *Workspace* for temporary calculations. Default values of 50 *Variables* and 1000 *Elements* are displayed after PLOT is run. *Workspace* sufficient for up to 250,000 values is available to each user. After *Workspace* has been allocated, the prompt:

```
ML>
```

appears. This is **home base**, from which individual programs are run.

To exit or quit *PLOT*, type *exit* or *quit* or press the CTRL key simultaneously with Z at the ML> prompt:

```
ML> exit
ML> quit
ML> CTRL Z
```

The *PLOT* programs can be used in three modes: 1) the **Interactive Mode** uses a system of menus and prompts; 2) the **Command Line Mode**, bypasses menus and prompts and inputs are typed in short command sequences; and 3) the **Batch Mode**, in which *PLOT* commands are executed from a text file. The interactive mode using menus is described first.

Individual programs are invoked from the ML> prompt using the program name, for example,

```
ML> ENTER
```

or using the program number as shown in Table 1, for example

```
ML> 1
```

Consistent with VMS convention, entries are not case sensitive, and HELP is available:

```
ML> help
```

Table 1 shows the HELP display which lists the basic *PLOT* programs. Appendix 1 provides a listing of the menus for these programs and summarizes *PLOT* applications. A study of Appendix 1 is required to understand *PLOT*'s capabilities, some of which will not be described in detail here.

Variable & Element Ranges: Number Lists

A key principle in using *PLOT* efficiently is understanding how to select a set of data for various data manipulations. The data set is identified by the *Variable* and *Element* number. Throughout PLOT the term *Variable* is used to indicate a column or vector of numerical data, all of the same type, all possessing the same *Variable Header* information. From time to time we may abbreviate a specific *Variable* number 12 as *Var#* 12. The term, *Element*, refers to a single data item in a specific *Variable*. To avoid wordiness, we may abbreviate *Element* 7 to *Elmt#* 7 or $N = 7$, and you will see that syntax in the data listings.

Thus a set of data is specified by entering a list of numbers giving the range of *Variables* and data *Elements*. In all operations the *Variables* (i.e. columns) are specified first and the data *Elements* (i.e. rows) second. Both *Variables* and *Elements* are specified by three parameters: 1) **starting index**, 2) **ending index** and 3) **interval**.

A *Number List* is shorthand notation to select *Variables* and *Elements*. Individual items in the *Number Lists* are separated by commas (,). Either the hyphen (-) or ellipses (..) is used to indicate a range of items between the first and second indices in either ascending or descending order. The colon (:) is used to indicate the interval. For example, 1-101:10 specifies *Element* #s 1,11,21...101. Unless otherwise specified, the interval is assumed to be one. Examples of *Number Lists* are shown in Table 2.

If the limit index is larger than the number of variables currently loaded or the number of data values currently in use, the prompt will be displayed again showing the actual number of *Variables* or data *Elements*. In some cases, you may wish to use all of

Table 1. The HELP display showing *PLOT* programs.

#	Program	Description
1	ENTER	Enter and Edit: Data, Scales, Captions, etc.
2	LIST	List: Data, Scales, Captions to Selected Device
3	FILE	Transfer Data: between *Archive* Files and *Workspace*
4	LOAD	Load Data: from *Archive* File to *Workspace*
5	RECORD	Record Data: from *Workspace* to *Archive* File
6	SIZE	Returns: *Workspace* or *Archive* File Size
7	GRAPH	Draw Graph: X-Y Line or Scatter Plots
8	TRANSFORM	Transform Variables: via MLHPL Algebraic Functions
9	MAP	Mapping: Mercator, Orthogonal, Satellite View
10	CONVERT	Convert Files: ASCII (Text) <--> MLML DBASE (Binary)
11	REGRESS	Regression: Polynomial, Multiple or Harmonic
12	STATS	Statistics: Univariate and Histogram
13	TIDE	Tidal Predictions: Daily High/Low or Fixed Time Interval
14	HEADER	Print or Edit series of File and Variable Headers
15	NUMER	Numerical Analysis: Integrate, Differentiate, Interpolate, Smooth, Spectral Analysis
16	TOOLS	Manipulate data: Transpose, Sort, Replace, Decimate, etc

the data *Elements*, but not know their number. The asterisk (*) may be used in place of a *Number List*. The asterisk indicates use of the default value or previously entered value. In the case of *Element* ranges, the asterisk indicates you wish to use all the data. Examples of this will be given later.

Menus

In the **Interactive Mode**, menus display program options. The ENTER menu, for example, lists 12 procedures (Appendix 1). After the menu is displayed, the prompt

ENTER: Enter Menu Choice

requests a selection of one of the program choices. This prompt can be answered in three ways:

1. Enter a valid menu choice (1-12) followed by RETURN

2. Enter 0 and press RETURN, or press CTRL Z

3. Press RETURN without typing a menu item

Action 1 causes the program to branch to that procedure. (Note that choices *cannot* be made by typing the description displayed in the menu.) Action 2 exits the current program and returns control to the main prompt ML>. Action 3 redisplays the menu. An invalid menu choice causes the menu to be redisplayed. Menu choices of 0 are used in all programs to return control to **home base** (ML>).

To exit PLOT, press CTRL Z and type *exit* or *quit* at the ML> prompt. You will then be given the chance of saving *Workspace Variables* if you haven't already done so.

It is worth noting here that in all PLOT programs the prompt indicates the current program. When you are in the GRAPH program, its prompt will display

GRAPH: Enter Menu Choice

the FILE program prompts begin with

FILE: Enter Menu Choice

and so forth.

Some PLOT programs may contain subprograms. Those within GRAPH such as GRAPH_PARAM, will indicate you are within a subprogram with prompts like

GRAPH PARAM: Enter Menu Choice

If you become hopelessly lost, you can return to ML> **home base** by pressing CTRL Z repeatedly or by exiting each menu by pressing 0 and RETURN.

Table 2. Examples showing use of Number List syntax.

Prompt	Number List	Result
ENTER Range of Var #'s to List:	1,7,5	List data for the three *Variables* 1, 7, 5 in that order
ENTER Range of Var #'s to List:	1-3,23,24	List values corresponding to data *Elements* 1, 2, 3, 23, 24
ENTER Range of Elmt #'s to List:	1..31:5,33,54	List values corresponding to data *Elements* 1, 6, 11, 16 , 21, 26, 31, 33, 54
ENTER Range of Elmt #'s to List:	100..50:10	List values corresponding to data *Elements* 100, 90, 80, 70, 60, 50
ENTER Range of Elmt #'s to List:	*	List values corresponding to data *Elements* 100, 90, 80, 70, 60, 50 as previously executed

Entering Data

Data may be entered into *PLOT* by several methods. A few values may be entered manually through the *ENTER* program. Space delimited data files may be made using a text editor or a spreadsheet program and read into *PLOT* using the *CONVERT* program.

Direct keyboard data entry is provided by the *ENTER* program. Data are entered for a single *Element #* in a *Variable* following single line prompts. This method is useful for small data sets. Header information including symbol, description, plot caption and scaling parameters may also be entered via other *ENTER* routines (Appendix 1). Examples of simple data entry are

```
*ENTER: Enter Data [1,1]:                    12.34
*ENTER: Enter Data [1,2]:                      22
```

and so forth... Note that the first index is the *Variable #* and the second index is the *Element #*. These indices update as you continue entering data. Data are only added following the last *Element #*. To *CHANGE* existing values, *INSERT* more values, or *DELETE* old values you must use those appropriate functions from the *ENTER* menu.

It is important in some situations to signify missing values. An IEEE standard for representing missing values called "Not a Number", or NaN is used in *PLOT* and some commercial software. The specific values for missing REAL*4 and REAL*8 data are shown in Appendix 2. When data are entered from the keyboard, missing values are expressed by pressing RETURN without typing a value. Thus

```
*ENTER: Enter Data [1,3]:                    RETURN
*ENTER: Enter Data [1,4]:
```

shows the display following entry of a missing value for *Var#* 1, *Elmt#* 3. Missing values or NaN are treated by all *PLOT* programs just as that: the average of many NaN's is undefined or NaN.

Table 3. MLHPL Functions used in *TRANSFORM* and the *MLHPL* calculator.

ABS	ACS	ADD	AIRMASS	ASN
ATG	ATN	BAND	BIT	CALDATE
CFM	CMF	CMP	COND	COS
COS	CSH	DEBUG	DEG	DELTA
DENSITY	DEPTH	DRND	DTO	EOR
ERL	ERN	EXIT	EXP	FLG
FLT	FRC	FREEZE	FXD	GCDIST
GCHEAD	HELP	IF	INT	IOR
JULDATE	JULDAY	LABSAL	LN	LOG
MAX	MDEC	MIN	MOCT	MOD
NOON	OTD	OXYSOL	PI	PRNM
PRTFN	PRTSV	QUIT	RAD	RES
RND	ROM	ROT	RTIME	SALIN
SFG	SGN	SHF	SIGMAT	SIN
SINH	SOUND	SPCHEAT	SQRT	SUNALT
SUNAZ	SUNRISE	SUNSET	TAN	TANH
THETA	THETAP	TN	TODEG	TODMS
UNIT	+	-	*	/
^				

The ENTER program incorporates the *MLHPL* command line algebraic interpreter. When the interpreter is in operation, the prompt line is flagged by an asterisk (*). Other entry points throughout the PLOT system also provide the interpreter to allow calculations during data entry. All functions available to the *TRANSFORM* program (Table 3) may be used. For example,

```
*ENTER: Enter Data [3,12]:                    pi*5^2
```

places the value of 78.539... (correct to 7 significant digits for REAL*4 precision or to 15 significant digits for REAL*8 precision) into *Var#* 3, *Elmt#* 12. The availability of command line algebra can be a real time-saver. *MLHPL* uses its own variables A..Z, whose values are retained from one program to another, and these *simple variables* may be used while entering results from complicated functions. For example, if you wish to enter the seawater density for the same salinity, but different temperatures you may execute the following

```
*ENTER: Enter Data [4,1]: 33.4}s;1.5}t;4500}p;density(s,t,p)
*ENTER: Enter Data [4,2]:          2.34}t;density(s,t,p)
*ENTER: Enter Data [4,3]:          3.45}t;density(s,t,p)
```

The three values stored into *Var#* 4, *Elmt#s* 1..3 are 1.047031, 1.046875 and 1.046660, respectively, and the temperature, salinity and pressure values are retained in *MLHPL* variables S,T,P. Note that the assignment operator is not the = symbol, but the right curly bracket }. In HPL this was called the *guzinta* operator, as in 4 *guzinta* A, which is typed as 4 } A. Note also that MLHPL simple variable names and functions are not case sensitive, consistent with VMS. HPL variables are stored in a file located in the user's SYS$SCRATCH directory, which by default is the users top level directory. Placing the MLHPL simple variables in a file makes them available to all programs that use the HPL interpreter.

It is possible to use *MLHPL* in the *ENTER* program as a scratch pad that returns the results of calculations without assigning results to a *PLOT Variable*. To do this, place an exclamation mark before the MLHPL expression. The following example shows how the display format can be set using the *fxd* (fixed decimal as opposed to *flt* for floating point or exponential format) function to set the display to 6 digits to the right of the decimal, then calculate seawater density for a given temperature, salinity and pressure:

```
*ENTER: Enter Data [4,1]:                     ! fxd5
*ENTER: Enter Data [4,1]:! 33.4}s1.5}t;4500}p;density(s,t,p)
1.04703
*ENTER: Enter Data [4,1]:
```

Note that the *Variable* and *Element* indices do not change because the exclamation mark prevents the result from being saved into the database. If results are returned, they are displayed on the following line and the prompt repeats the request to enter data for the current *Variable* index. The above example also shows that long expressions can wrap across lines. Up to 512 characters are allowed in MLHPL lines, and multiple expressions are separated by the semi-colon.

To review the contents of the MLHPL simple variables, or to "print simple variables" type

```
*ENTER: Enter Data [4,1]:                     !prtsv
```

To review the MLHPL function names or to "print functions", type

```
*ENTER: Enter Data [4,1]:                     !prtfn
```

To obtain help with MLHPL functions, type

```
*ENTER: Enter Data [4,1]:                     !help
```

then page through the HELP library looking for help with specific FUNCTIONS_AND_COMMANDS.

For large amounts of data, it may be more efficient to import data using ASCII (American Standard Code for Information Interchange) files of numerical columns and rows following the format shown in Table 4. You may generate these files using all editors, word processing programs and spreadsheets that produce ASCII plain text files. Files like that shown in Table 4 are generated by both the VMS and HPL *PLOT* programs and can be imported directly into VMS *PLOT*.

None of the *Header* information following a leading exclamation mark is required, and those fields need not be entered (which is likely to be the case when you generate ASCII files outside of *PLOT*.) In that case, the ASCII file will look like the line labelled "Numerical Data Array". However, if you do want to enter the header information by way of ASCII files, that format must be followed strictly. Data without header information can be imported from ASCII files made up of column and row data, with individual data elements separated by spaces. Data *Elements* within a single column are interpreted as belonging to a single *Variable*. Any number of VAX, MS-DOS, or Macintosh programs provide this capability. The *CONVERT* program described later imports ASCII files into *PLOT*.

An efficient way to enter large amounts of data is by way of computer data acquisition systems using programs that store results in *PLOT* compatible files. The MLML physical oceanography group uses several such data acquisition programs (NOAA radiometers: MOS, SIS, FASTIE; the MLML CTD and long-track systems) to directly generate *PLOT* binary files whose structure is described later. It is also a relatively simple task for a data acquisition system to generate ASCII column and row files that can be read by the *CONVERT* program.

Listing Data

All of the values and variable headers in a PLOT data file can be printed either on the terminal, on a hardcopy printer, or to a file. The LIST menu options (Appendix 1) allow you to list the numeric data, the graph scaling parameters and captions (described later), the *Variable*'s symbol, listing *Format*, and finally the *File Header* and *Variable Description*. The listing *Format* controls the appearance of how the values are printed. The format uses FORTRAN syntax. For example 123.4567 is displayed in the following ways depending on the *Format*

Format	Appearance
F11.3	___123.457
F8.2	__123.46
F5.0	__123
E10.3	__0.123E03
1PE10.3	__1.235E02

where F means fixed decimal point, E exponential notation where the number left of the decimal point gives the number of columns, and the number to the right gives the number of digits displayed right of the decimal point. The 1P format prints the one digit to the left of the decimal point. Results are displayed with correct rounding. The listing format does not change the precision of the value as stored in the *Variable*. Until the *Format* is set by the user, all *Variables* are listed using the default F11.3.

Each *Variable* has its own listing *Format*, which can be entered or changed in the *ENTER* and *LIST* programs. If a *Variable* contains a *Symbol*, it will be printed as a column header, otherwise Var#1, Var#2, and so forth are default column headers.

Data listings are directed to the user's terminal, to a hardcopy device or to a file through the LIST menu selection 6. The default setting is *SYS$OUTPUT* which is the VMS logical for the user's terminal. This can be replaced by any VMS file name or any of the hardcopy devices displayed in the LIST submenu and some devices that are not displayed there.

The following example shows most of a listing's features for *Variables* #1 and #2:

```
File: TEST_DATA.MLDAT

 N   Weight      Var#2
 1     0.29      0.085
 2     1.15      0.332
 3     2.60        ***
 4     4.63      1.344
 5     7.24      2.100
```

In addition to the *Format* parameter discussed above, two other parameters control the appearance of a data listing. *LIST* Menu Selection 7 allows the user to suppress printing of the *Element* #s in the first column. It also allows replacement of the blank spaces, which are the default case for missing values, by any printable characters. Your choice may be dictated by what is appropriate for other programs. In the example above, missing values are shown as the characters, ***. For external programs that expect numeric characters in all fields, 999 might be used to signify missing values.

In the above example, a *Symbol* is used for *Var#* 1, but not for *Var#* 2; the missing datum in *Var#* 2, *Elmt#* 3 were printed as ***; the *Format* for *Var#* 1 is F7.2 and that for *Var#* 2 is the default F11.3; printing of *Elmt#*s in the first column was not disabled; the file name is always written as the first line of a listing. Before printing out a hardcopy, it is wise to test list the data to your terminal.

Transforming Data

The TRANSFORM program is used to perform arithmetic among *Variables*, such as adding *Var#* 1 and *Var#* 2, and placing the results in *Var#* 11. This is done as follows:

```
TRANSFORM: Enter Expression:                          X + Y
TRANSFORM: Enter Var# to use as X:                        1
TRANSFORM: Enter Var# to use as Y:                        2
TRANSFORM: Enter Var# to save Transformed X data:        11
TRANSFORM: Enter Var# to save Transformed Y data:         0
TRANSFORM: Enter Range of Element #:                    1-4
```

The results are shown by listing *Var#*s 1,2, and 11:

```
N   Var#1   Var#2   Var#11
1   1.00    2.00    3.00
2   1.11    2.22    3.33
3   2.34    3.42    5.76
4   0.44    6.66    7.10
```

Notice that the transform syntax does not use an assignment operator. The algebraic expression is coded using dummy variables X and Y (not to be confused with PLOT *Variables* or MLHPL simple variables). The results are assigned to the PLOT *Variable* 11 as indicated by the fourth prompt above. A future PLOT revision will allow use of *Symbols* to replace the dummy variables (X and Y in this example).

It is frequently useful to use the sequence numbers, N, to form values without the use of PLOT *Variables*. Here N takes the value of the *Element #* specified in the *Element* range prompt. The following TRANSFORM examples illustrate this:

```
TRANSFORM: Enter Transform Expression:                N
TRANSFORM: Enter Var# to save Transformed X data: 12
TRANSFORM: Enter Range of Element #:                  1-5
Transforming ...

TRANSFORM: Enter Expression:                          N ^2
TRANSFORM: Enter Var# to save Transformed X data: 13
TRANSFORM: Enter Range of Element #:                  1-5
Transforming ...

TRANSFORM: Enter Expression:                          SQRT(2*(N-1))
TRANSFORM: Enter Var# to save Transformed X data: 14
TRANSFORM: Enter Range of Element #:                  1-5
Transforming ...
```

with results:

```
N   Var#12  Var#13  Var#14
1   1.000    1.000  0.000
2   2.000    4.000  1.414
3   3.000    9.000  2.000
4   4.000   16.000  2.449
5   5.000   25.000  2.828
```

The *Element* number, N, can be used in place of a "for/next" loop to generate a variety of numerical series. The three examples above might be coded in BASIC as

```
FOR I = 1 TO 5 BY 1
    PRINT I,I^2,SQRT(2*(I-1)
NEXT I
```

Because values are stored in consecutive *Elmt#*s, performing for/next loops with fractional numbers might be done in BASIC as follows

```
FOR XX = -0.11 to -0.55 BY -0.11
    PRINT XX
NEXT XX
```

This would be done in TRANSFORM as

```
TRANSFORM: Enter Expression                           -0.11*N
TRANSFORM: Enter Var# to save Transformed X data: 15
TRANSFORM: Enter Range of Element #:                  1-5
Transforming ...
```

Conditional expressions and multiple statements are illustrated by the following

```
TRANSFORM: Enter Expression:                  X/2;if N>3;X/4
TRANSFORM: Enter Var# to use as X:                       12
TRANSFORM: Enter Var# to save Transformed X data: 16
TRANSFORM: Enter Range of Element #:                  1-5
Transforming ...
```

Another important procedure is to generate the missing value code (NaN) for specific values contained in *Variables*. This is done somewhat cryptically in HPL code

```
TRANSFORM: Enter Expression:    nan;if X>2;if X<10;X
TRANSFORM: Enter Var# to use as X:                   13
TRANSFORM: Enter Var# to save Transformed X data: 17
TRANSFORM: Enter Range of Element #:                 1-5
Transforming ...
```

The last example **does not** replace *Elements* in *Var#* 13 whose values are greater than 2, but less than 10. That is, the expression returns the original value X. Values outside that range are replaced by the missing value code *NaN*. Listings from the last three transforms are

```
N   Var#15  Var#16  Var#17
1   -.110    0.500
2   -.220    1.000  4.000
3   -.330    1.500  9.000
4   -.440    1.000
5   -.550    1.250
```

Because fairly sophisticated expressions may be evaluated in *TRANSFORM*, it is useful to save these statements in text files. Any of the above *TRANSFORM* expressions may be entered into a file (having a default extension of .HPL). The following file named AOU.HPL illustrates the use of HPL simple variables and computes Apparent Oxygen Utilization (μmoles/kg) from *Variables* containing dissolved oxygen (ml/liter), salinity (PSU), and temperature (C):

```
X } O
Y } T
Z } S
DENSITY(S,T,0)}D
OXYSOL(S,T)}B
(B - O)*1000*D/22.39
```

This HPL file is executed in *TRANSFORM* as follows

```
TRANSFORM: Enter Transform Expression:                @AOU
TRANSFORM: Enter Var# to use as X:                        1
TRANSFORM: Enter Var# to use as Y:                        2
TRANSFORM: Enter Var# to use as Z:                        3
TRANSFORM: Enter Var# to save Transformed X data:         4
```

```
TRANSFORM: Enter Var# to save Transformed Y data: 0
TRANSFORM: Enter Var# to save Transformed Z data: 0
TRANSFORM: Enter Range of Element #:              *
Transforming ...
```

The following shows the results for the oxygen, salinity and temperature values in *Var#*s 1-3.

Var#1	Var#2	Var#3	Var#4
6.780	33.330	5.000	-33.114
5.670	34.440	10.000	30.765
1.230	35.550	15.000	203.495

Note that the asterisk caused the expressions to be computed for all of the data *Elements* in *Var#* 1.

TRANSFORM expressions may be executed from the ML> prompt as described later.

Data Files: Loading and Recording

The *PLOT* program, *FILE,* and its subprograms, *LOAD* and *RECORD*, retrieve and store the *Workspace* contents on disk. Terminology dealing with files is explained in some detail later.

LOAD retrieves data from *Archive* disk files into the *Workspace* from which the data are processed, graphed, listed, and so forth. *LOAD* is accessed through *FILE* menu choice 1 to load complete *Variables* including scale parameters. As with other operations, the *Number List* routines are employed:

```
FILE:  Enter Menu Choice:                          1
LOAD File Name:                                    pl314r
LOAD Range of Variable #'s from Archive File:      1-19
LOAD Starting at WORKSPACE Var #:                  1
```

Type in the complete VMS file name with or without the file extension, .MLDAT, which identifies an MLML_DBASE binary file. ASCII or plain text files cannot be loaded from this menu but are accessed through the *CONVERT* program.

Often you will want to load all disk variables to *Workspace*, and you may not know the number of *Variables* in the disk file. If *PLOT* finds the file, it automatically displays the number of *Variables* after the range of variables prompt. Pressing RETURN without changing the *Variable* range displayed causes all file *Variables* to be loaded. By default, *Variables* are loaded sequentially into *Workspace* starting with *Variable* 1. You may change the *Archive* File from which *Variables* are loaded as well as the starting *Workspace Variable* to which they are loaded. As the file is loaded, the *File Header* and *Variable Descriptions* will scroll down the screen.

RECORD saves complete *Variables* from your *Workspace* to an *Archive* disk file. As explained further in a later section, the term *Archive* file is used to indicate permanence of these files. VMS always creates a new version of a file, and each time you *RECORD* a file, a new version is created, leaving the original intact. The recording dialogue is similar to that for loading:

```
FILE:  Enter Menu Choice:                          6
RECORD File Name:      [DATA.NOAA]PL314R.MLDAT
RECORD Range of Variable #'s from Workspace:       1-22

Enter Identifier # 1
CRUISE: CZCS Post-Launch 3
RETURN
Enter Identifier # 2
Station 14:  Western Sargasso
CTRL Z
```

At this point you are prompted to enter *File Header* strings called Identifier #1...#8. These 80-character strings hold information describing the contents of the file, not the individual *Variables*. Use of these "header strings" is at the user's discretion, but good annotation may be the difference between valuable data and digital junk. The same can be said for the *Variable Descriptions*. To terminate entry of the *File Headers*, press CTRL Z. Prompts for the remaining headers will be bypassed, and the file will be recorded including all 8 of the header strings.

The *FILE* program provides the means to load and record complete files, only *File Headers*, only *Variable Headers*, or only the numerical data. Each of these operations leaves intact other components of the file.

At times, it is useful to append one numerical data column to another. This may be done with the *FILE* menu option *CHAIN*. The syntax is similar to loading, and this results in *Archive Variables* appended following the last *Element #* of the specified *Workspace Variables*.

Because loading and recording are such common tasks, *PLOT* provides shortcuts directly to the *LOAD* and *RECORD* procedures. Thus typing *LOAD* or *RECORD* from the ML> prompt immediately displays the loading and recording prompts described above. Loading and recording may be done completely on the command line without prompts as explained later.

Producing a Graph

X-Y line and scatter graphs are generated from the *GRAPH* program via its menu and submenus

(Appendix 1). Since *PLOT* can plot on one of several graphics devices, the user must select a specific device. After a hardcopy plotter or printer has been used, ALL output is directed to that device. After a hardcopy graph has been made, it may be wise to immediately change the graphics device to the user's terminal. The normal procedure is:

1. Select the graphics device and port (by default Tektronics 4014 graphics on your terminal, or by choice one of a number of hardcopy plotters and printers).

2. If necessary, clear the display or eject the page from a hardcopy device.

3. Label and outline the graph.

4. Select the plot type and other parameters controlling its appearance.

5. Plot the data.

6. Change the graphics device back to your terminal.

Assuming you will be plotting to your terminal using default graph parameters, the dialogue from the *GRAPH* menu is

```
GRAPH: Enter Menu Choice:                    1 (clear screen)
GRAPH: Enter Menu Choice:                    2 (label & outline)
GRAPH: Enter Variable # for X axis:          1
GRAPH: Enter Variable # for Y axis:          2
```

At this point the axes to the graph with captions and labels will be drawn on your display.

Next you specify the range of *Element* #s to be plotted. By default, the entire data range will be displayed after the prompt. You may choose to use all of the data, or you may select a portion to be plotted using *Number List* syntax.

```
GRAPH: Enter Menu Choice:                    3 (plot data)
GRAPH: Enter Variable # for X axis:          1 (default)
GRAPH: Enter Variable # for Y axis:          2 (default)
GRAPH: Enter Range Element # for Graph:      1-80 (default)
```

At this point the X-Y data will be plotted.

Some PC and Macintosh graphics programs require the user to "hold" the previous graph to permit overplotting. *PLOT* operates in just the opposite fashion, and overplotting is done by default until you *CLEAR* the display or eject the page. After viewing the graph press RETURN to return to the program prompts. To overplot another line, select *GRAPH* menu choice 3 and specify another X-Y data pair. Note that the axes are labelled according to the first X-Y *Variables* used to outline and label by *GRAPH* menu choice 2. Subsequent X-Y graphs are scaled according to current scale parameters WHICH NEED NOT BE THE SAME as those used to initially outline and label. This feature is useful for reviewing quickly data in different *Variables*, but for published graphs, all plotted data should use the same axis scales.

Multiple *Variables* may be plotted in one operation by specifying a range of X or Y variables. The following shows how three lines are plotted using every tenth data *Element*.

```
GRAPH: Enter Menu Choice:                    3
Enter Variable # for X axis:                 1
Enter Variable # for Y axis:                 2-4
Enter Range Element # for Graph:             10-80:10
```

Elements of a Graph

One feature of *PLOT Variables* is that the numerical data are accompanied by parameters used to format line graphs. The data structure is defined explicitly in Appendix 2 in the STRUCTURE called /MLML_VAR_HEADER/. The application of these scale parameters is illustrated in Figure 1. The *Caption* should be differentiated from the *Description*. The latter describes the contents or data processing concerning the numeric data, while the former contains succinct text written on the X or Y axes.

The following shows the menu for setting *Scale Parameters* that define the properties of the graph shown in Figure 1.

For the X *Variable*:

```
GRAPH: Enter Menu Choice:                                  5
GRAPH SCALES: Enter Range of Variable #'s:                 1-2
GRAPH  SCALES:  1=Capt&Scales  2=w/GraphExt
    3=Captonly                                             2
GRAPH: Enter Caption for Var #  1:          Wavelength (nm)
*GRAPH: Enter Scale Min:                                   300
*GRAPH: Enter Scale Max:                                   700
*GRAPH: Enter Tick Interval:                               25
*GRAPH: Enter Ticks/Label:                                 4
GRAPH: Enter Decimals (0 to 9):                            0
GRAPH: Enter Axis Type (?=list types)                      ?
Available Axis (Tick Spacing) Types:
    1 = Linear
    2 = Log (base 10)
    3 = Log (base e)
GRAPH: Enter Axis Type (?=list types)                      1

GRAPH: Enter Label Type (?=list types)                     ?
```

```
Available Numeric Label Types:

1 = Fixed Decimals
2 = Fixed with Floating Decimals
3 = Float (Scientific Notation)
4 = 10^(power)
5 = e^(power)
6 = Month
7 = Decimal Degrees
8 = Degrees and Decimal Minutes
9 = Degrees, Minutes and Decimal Seconds
10 = Decimal Hours
11 = Hours and Decimal Minutes
12 = Hours, Minutes and Decimal Seconds

 GRAPH: Enter Labelling Choice:                1
*GRAPH: Enter Label Offset from Min Scale:     4
*GRAPH: Enter Label Offset from Max Scale:     4
*GRAPH: Enter Ticks per Grid Line:             0
*GRAPH: Enter Grid Density (dots/inch):        0
```

Scale Min and *Scale Max* define the plotting limits. *Tick Interval* sets the spacing between tick marks; *Ticks/label* set the frequency that the labels (the numeric values) are written and *Decimals* sets the number of digits written to the right of the decimal point. It is possible (but not shown here) for *Scale Max* to be less than *Scale Min*, in which case the *Tick Interval* is negative. The first four *Scale Parameters* are required. For best results, these values should be selected by the user, but the *AUTOSCALE* menu choice in GRAPH will set them based on the data range in the selected *Variable*. Note that the MLHPL interpreter is active at these prompts as indicated by the asterisk (*). As a matter of style, we point out: "Too many labels spoil the plot".

The next six *Scale Parameters* may be entered optionally and are selected when you choose to enter *2=w/Graph Ext*. The X axis in the example is a simple linear *Axis Type* with *Linear* label spacing, but it does show that the labels may start to the right of the Scale Min (*Label Offset from Min*), and end to the left of Scale Max (*Label Offset from Max*). The Y axis uses a log (base10) *Axis Type* and the *Label Type* is set to 10^power. Notice several *Label Types* are available for maps using degrees and minutes labels, and time series plots with hour or month labels. The gridding option extends the tick marks from *Scale Min* to *Scale Max*.

The Y axis is setup similarly, but here the menus for the *Axis Types* and *Label* For some of the labels, the *Decimal* setting has no effect. When months are selected the *Decimal* setting specifies the number of characters used in labelling the month (3 results in Jan, Feb, Mar...).

```
GRAPH: Enter Caption for Var # 2: Lw(uW/cm^2/sr/nm)
*GRAPH: Enter Scale Min:                      0.0001
*GRAPH: Enter Scale Max:                      10
*GRAPH: Enter Tick Interval:                  0.2
*GRAPH: Enter Ticks/Label:                    5
 GRAPH: Enter Decimals (0 to 9):              0

GRAPH: Enter Axis Type:                       2
```

Notice that the *Scale Min* and *Scale Max* values for log scaled axes are in arithmetic units while the *Tick Interval* is specified in log units. Thus the *Tick Interval* of 0.2 results in 5 tick marks for each log unit between 0.0001 and 0.001, etc.

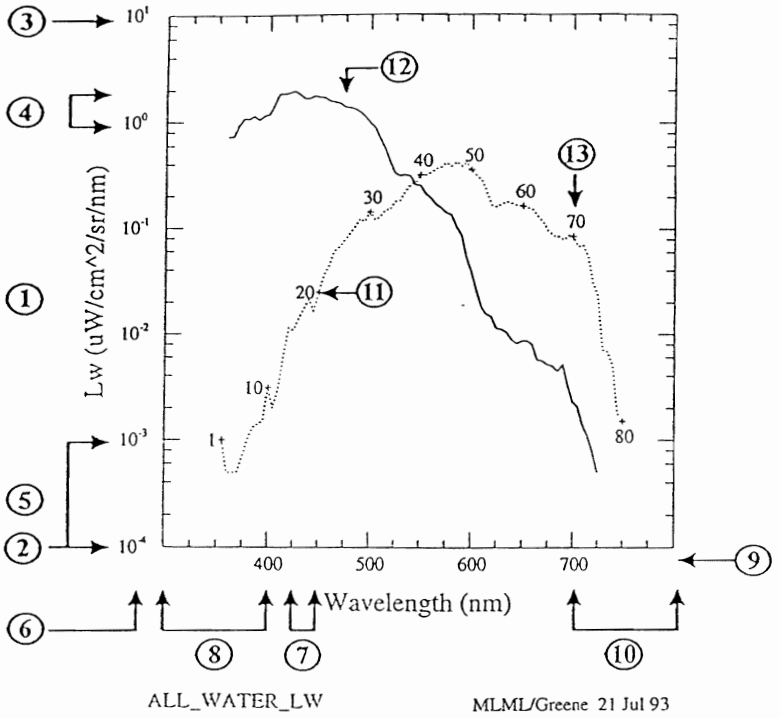
```
 GRAPH: Enter Labelling  Choice:              4
*GRAPH: Enter Label Offset from Min Scale:    0
*GRAPH: Enter Label Offset from Max Scale:    0
*GRAPH: Enter Grid Ticks:                     0
*GRAPH: Enter Grid Density:                   0
```

Other characteristics of the graph are controlled from GRAPH menu choice 5: *SELECT Graphics Parameters*. Unlike the *Scale Parameters* which follow the *Variables* whenever they are moved or recorded, the *Graphics Parameters* are associated with the entire graph, not with a particular *Variable.* Because these graphics features are complex, they are saved in a binary file (by default the last parameters used are in GRAPHICS_PARAMETERS.DAT) and these will be automatically retrieved for use during the next session. The current settings may also be recorded in a user-named file.

To make a straight forward line graph, select *Plot Type* = 1; with "l" (lower case L) *Draw Line Between Points*; and *Line Type* = 1 for a solid line. These are the defaults. The following menus show the wide range of plots that may be selected.

```
GRAPH:  Enter Menu Choice:                    7
Enter Graphics Parameters:
  1 = Plot Type and Sizes
  2 = X Axis
  3 = Y Axis
  4 = Z Axis
  5 = Font and Outline
  6 = Viewport
  7 = File/Date/Time Stamps
  8 = Display Current Settings
  9 = Reset to Default Settings
 10 = Load from File
 11 = Record to File

GRAPH PARAM:  Enter Menu Choice               1
```



Scale Parameters

	Y Axis:			X Axis:	
(1)	Caption:	Lw (μW/cm^2 sr nm)		Caption:	Wavelength (nm)
(2)	Scale Minimum:	0.0001		Scale Minimum:	300
(3)	Scale Maximum:	10		Scale Maximum:	800
(4)	Tick Interval:	0.2	(7)	Tick Interval:	25
(5)	Ticks/Label:	5	(8)	Ticks/Label:	4
	Decimals:	0		Decimals:	0
	Axis Type:	Log base 10		Axis Type:	Linear
(6)	Label Type:	10^power	(9)	Label Type:	Fixed decimal
	Offset from Minimum:	0		Offset from Minimum:	4
	Offset from Maximum:	0	(10)	Offset from Maximum:	4

Graphics Parameters

(11)	Plot Type:	'+'
	Plot Type Size:	2 (% of Y axis)
(12)	Plot Type:	'l' (solid line)
	Line Type:	1

			Caption			Label	
(13)	Plot Type:	'n'	Position	Size	Rotation	Size	Rotation
	Z Label Position:	top	X Axis: Bottom	6	0	4.5	0
	Z Label Decimals:	0	Y Axis: Left	6	90	4.5	0

Figure 1. A *PLOT* graph illustrating use of the *Scale Parameters,* which accompany each *Variable* and some of the *Graphics Parameters*, which do not.

```
GRAPH PARAM: Enter Plot Type or Symbol (? = list)   ?

     Available Plot Types:

     l = Draw Line Between Points
     m = Plot GKS Markers
     n = Label Element Numbers
     p = Plot Points (The period (.) does the same)
     z = Label Z (third Variable) Values

Symbols "+", "*", "O" and "X" are GKS markers.
Other characters are plotted as a symbol.

GRAPH PARAM:  Enter Plot Type or Symbol (? =list): l
```

When a line graph (lower case L) is chosen, a list of available lines is optionally displayed.

```
*GRAPH PARAM:  Enter Line Type  (? = list types):    ?

Available GKS Line Types:
     1 = Solid
     2 = Dashed
     3 = Dotted
```

The example in Fig. 1 shows three different plot types. The solid line uses line type 1, and the dotted line is line type 3. The dotted line was overplotted using the "+" symbol, followed by an "n" plot to label each X-Y data pair with the *Element #* for data pairs N = 1,10..80:10.

Further control over the graph's appearance is provided through the *X,Y,Z Axis GRAPHICS Parameter* choices. These allow the *Captions* and *Labels* to be written in four orientations with different character sizes. The *Viewport* settings control where single or multiple graphs will be drawn on the screen or printed page. The graph can also be annotated with filename, time and date stamps. With as many parameters that are possible, the *Display Current Settings* selection is useful to understand the range of possible graphics control, and *Reset to Default Settings* is a convenient way to return to a simple setup. You may *Record to File* the current graphics setup and later *Load from File* those settings.

File Conversion

The *CONVERT* program is used to change *PLOT* binary files into editable ASCII text files and vice versa. The program works in four ways (Appendix 1): ASCII text files on disk can be directly converted to *PLOT Archive* binary files without going through workspace. What this means is that no matter the size of the original ASCII file, a *PLOT Archive* file will be made.

```
CONVERT: Enter Name of Text File to Convert
                              [smith.data]november_data.txt

CONVERT: Enter Name of MLML Dbase File to Save Data
                                              [smith.data]nov87
Loading Header...
Text File NOT in Standard MLML Format
A temporary memory space is required to load the text file.
Enter the minimum number of variables or columns and the
minimum number of data elements or rows contained in the
text file.

CONVERT: Enter Number of Variables:                        10
CONVERT: Enter Number of Elements:                         55

Loading Data, Elmt #
...
Recording header...
Recording Var # 1...

Recorded File: DUA1:[SMITH.DATA]NOV87.MLDAT;1
7 Variables,   48 Elements
Date created: 26-JUL-93
```

In the second method, *CONVERT* translates the text file into *Workspace Variables*. In this way the user has immediate access to the data with all *PLOT* programs, but the *Workspace* must have been dimensioned sufficiently large to hold the data.

The reverse processes are conversion of *PLOT* binary files to ASCII from either disk *Archive* files or from *Workspace*. The dialogue for conversion from *Workspace* to ASCII follows:

```
CONVERT:  Enter Menu Choice                                4
Enter Range of Workspace Variables #s to Convert         7-11
Enter Name of File to Save Data
                                                       NOV87

Recorded File:
DUA1:[SMITH.DATA]NOV87.DAT;1 from Workspace
5 Variables,  48 Data Elements
From Workspace Variables 7-11
```

The procedure to convert binary *Archive* files to ASCII files works similarly. An ASCII file produced by this process is shown in Table 4. Note that data are delimited by spaces (ASCII character 32).

Command Line Mode

PLOT programs allow the user to bypass menus to specify the program function, the range of variables, and data elements from the ML> command line or from a batch file. The main program, itself, can be called from the VMS $ prompt by passing the number of *Variables* and *Elements* from the ML> command line. The following line starts *PLOT*

```
$ PLOT 25 2000
```

and allocates memory for 25 *Variables* each with 2000

Table 4. Example of ASCII data file in MLML_DBASE format.

```
                    ! File: GEO202
                    !  6 Variables, 15 Data Points
                    !
File Identifiers    ! CRUISE: GEOSECS PACIFIC OCEAN,  Leg  1
                    ! STATION: 202
                    ! Date: 30 Aug 1973     Position:  33*06.0 N  139*34.3 W
                    ! Bottom Depth: 4999
                    ! GEOSECS/TTO Data from Lamont-Doherty Geological Observatory
                    ! Extracted from 9-Track Tape, Reaves, 27 Jan 1987
                    ! Rearranged variables from Lamont's listing to MLML CTD format
                    !
Creation Date       ! Recorded: 10:20:23   11 Apr 1988
                    !
File Header Auxiliary Array
                    ! Aux:   0  0 33.06 139.3432  0 26905  0  0  0  0  0  0  0 4999  0
                    !
Variable Description
                    ! Col  1: Salinity (psu)
                    ! Col  2: Temperature (deg C)
                    ! Col  3: Depth (m)
                    ! Col  4: Oxygen (µM/kg)
                    ! Col  5: Bottle Number
                    ! Col  6: Sigma-theta
                    !
                    !          Min  Max  Tck  T/L  Dec  Aux  Fmt  Typ
Graph Scaling Parameters
                    ! Scl  1:    32   36    1    1    0    5  8.3    0
                    ! Scl  2:     0   30    5    1    0    2  7.2    0
                    ! Scl  3:  6000    0 -1000   1    0    3  6.0    0
                    ! Scl  4:     0  300   50    2    0    4  6.0    0
                    ! Scl  5:     0 10000 1000   2    0    0  5.0    0
                    ! Scl  6:    20   30    1    2    0   13  7.2    0
                    !
Numerical Data Array
 6 Variables x 15 Elements
                      34.646  21.77     0   999 7201 999.00
                      34.580  18.72    47   257  702  24.77
                      34.572  16.34   102   249  704  25.35
                      34.149  11.63   202   231  708  26.00
                      34.103   9.41   311   214  711  26.36
                      34.006   7.86   401   180  715  26.52
                      33.968   5.62   502   122  716  26.79
                      34.188   4.70   697    23  720  27.07
                      34.286   4.20   820    11  722  27.20
                      34.341   3.96   896    10  723  27.27
                      34.407   3.49  1030    13  402  27.37
                      34.548   2.60  1521    44  407  27.57
                      34.612   1.95  2023    74  412  27.68
                      34.663   1.54  3007   119 1011  27.75
                      34.681   1.48  4091   149  108  27.78
```

Elements. Because *PLOT* can use data types other than the default, MLML_DT_REAL, with 7 significant digits, the program can be started from the VMS prompt stating other data types. For example,

```
$ PLOT 30 1500 5
```

starts *PLOT* with 30 *Variables*, each containing 1500 double precision, 15 significant digit, values. The third parameter, 5, indicates the data type as MLML_DT_DBLE. Integers and complex data types can also be used in *PLOT* (See Appendix 2).

All *PLOT* programs can be run from the ML> prompt, thus avoiding the need to display each of the menu prompts. For example, the following line

```
ML> LIST 1 3-6,14 1-9999:10
```

runs the *LIST* program; selects menu choice 1 (print data values); prints every 10th data *Element* from the first to the last (presuming the *Variables* hold fewer than 9999 elements), for *Variables* 3, 4, 5, 6, 14. By stipulating *Elmt#* 1-9999, the program displays the actual range of *Elements*, 1-71:10. Notice that parameters on the command line are separated by spaces, but that the number lists are not. That is, the *Element Number List* was not entered

```
ML> LIST 1 3-6 14 1 - 9999: 10
```

because that would have been interpreted as three separate parameters by the number list routines. Entering values separated by spaces may be a natural way of entering expressions, and that may be done if the numbers are enclosed by double quotes.

```
ML> LIST 1 "3-6 14" "1 - 9999: 10"
```

Loading and Recording files between *Archive* and *Workspace* is eased by use of command line statements. First of all, you may execute any VMS commands by preceding them with the $ after the ML> prompt. To determine the names of MLDAT files in the current directory, type

```
ML> $ dir *.MLDAT

Directory DUA1:[SMITH.DATA]
ML_APR86.MLDAT;1  ML_APR87.MLDAT;3
ML_MAY86.MLDAT;2  ML_MAY87.MLDAT;1

Total of 4 files.
```

Based on the directory listing you then load all *Variables* from the selected file. Notice the use of the asterisk as a placeholder to use the default, which is all of the *Variables* contained in the file.

```
ML> load ml_may86 * 1
Loading Header ... ML_MAY86
Loading ... ML_MAY86
from      ARCHIVE   Var#s: 1-5
to        WORKSPACE Var#s: 1-5

#    File Header
1: Moss Landing Marine Labs Weather Data: 30-min avr
2: from 01 to 30 May 1986
3: Reaves, 02 Sep 86
...
7: New barometer added 02 Jan 86
Date created:20-Jul-86

Var# Symbol  Variable Description              #Elmts
1:   Day     Day of month (dec days May 86)     1440
2:   Time    Time of day (dec hours)            1440
3:   Wmag    Wind speed (mph)                  1440
4:   Wdir    Wind direction (deg true)         1440
5:   Baro    Barometric pressure (mbar)        1440
```

Suppose that after plotting the barometer data you discover that an offset appeared for *Elements* 631 through 854. You may correct them from the ML> prompt line as follows

```
ML> transform X+0.25 5 5 631-854
```

It is important to note that the algebraic expression

```
X + 0.25
```

has been entered without spaces. Because arguments on the command line are set apart by spaces, spaces cannot be placed between symbols in an algebraic expression. The following method also works by enclosing the algebraic expression including spaces inside quotation marks.

```
ML> transform "X + 0.25" 5 5 631-854
```

Next you wish to record the updated file contents back into a newly named *Archive* file. This can be done from the command line as

```
ML> record new_ml_may86 1-5

Enter Identifier # 1
1: Moss Landing Marine Labs Weather Data: 30-min avr
RETURN
2: from 01 to 30 May 1986
RETURN
3: Reaves, 02 Aug 86 changed barometer offset +.25
RETURN
4:
CTRL Z

Recording Header ...
Recording NEW_ML_MAY86
Recorded File:  NEW_ML_MAY86.MLDAT;1

from WORKSPACE Var#s: 1-5
to   ARCHIVE   Var#s: 1-5

#  File Header
1: Moss Landing Marine Labs...
...
Date Created 20-JUL-86
```

In this process, the *RECORD* program requested inputs for file headers 1 and 2, which were not changed. The third file header was changed by noting the barometer offset, and there was no need to change further file headers so CTRL Z bypassed those prompts, but recorded their original contents. The file creation date was not changed, but the VMS directory entry shows the date that this revised file was recorded.

The following repeats a previous example of a complex transform, but here it is shown as executed from the ML> command line.

```
ML> transform oxysol(X,Y)*1000/22.39/density(X,Y,0)-Z
      15 16 17 18 0 0 *
```

Note that three input *Variables* are required (15,16,17) and that one output *Variable* is saved (18). The menu requests possible, but unused output *Variables* for Y and Z (0,0) and the asterisk (*) replaces the default RETURN used in the menu mode to indicate use of all data *Elements*. Again, no spaces are allowed in the

algebraic expression.

After you have gained some confidence using the *PLOT* programs through their menus, you may find command line operations speed your work. All *PLOT* programs have the capability of passing parameters from the ML> prompt, however some programs may pause for some prompts.

Batch Mode

An important purpose of *PLOT* is to allow data processing to be automated. The user may find that the same PLOT procedures are repeated many times. In those cases it may be efficient to write a text file containing the keystrokes used to invoke the program and to select input and output *Variable* and *Element* ranges. The default file type used for these "scripts" is .MLCOM.

Batch files are called from the **home base** prompt using the ampersand similar to the way DCL command procedures are called in VMS:

```
ML> @STAT1_01.MLCOM
```

where STAT1_01.MLCOM is the name of a batch file in the user's default directory.

Table 5 shows three different batch files: the first uses the STATS program to save *Variable* (i.e. column-wise) statistics; the second uses the *TRANSFORM* program to apply a linear correction to selected *Variables*; and the third example applies a "Clipping" procedure in a *PLOT* program written to process NOAA spectroradiometer data. Note that the liberal use of comments (preceded by the exclamation mark, !) adds self-documentation to these .MLCOM files.

Terms Dealing with Files

The following is a more formal introduction to *PLOT* terminology pertaining to the file structure. Users of *PLOT* generally do not need to be concerned with the details required for programming, but knowledge of the following ideas is useful. Figure 2 illustrates files and the terminology, and Appendix 2 shows the data structure explicitly. This section will follow that organization by presenting a simplified explanation of the data, followed by a description based upon the explicit structure in Appendix 2.

The *PLOT* routines provide the interfaces between computer *Memory*, *Archive* files and *Workspace* files. Subroutines (described in Appendix 3) allocate computer memory to hold *Variables* temporarily while computations are done. The subprocess interface allows creation of a temporary *Workspace* disk file and corresponding logicals. The disk *Workspace* is used to transfer data between subprocesses, which are different *PLOT* programs operating on the same data set. The disk *Workspace* is also used to transfer data between the user's *Archive* files and computer *Memory*.

File

A *PLOT* file is a collection of a *File Header*, the *Variable Header*, and the *Variable* numeric data. Each *PLOT* file is referenced by a VMS filename and the file contains a variety of (binary) data.

The file is shown in Appendix 2 to consist of the FILE HEADER BLOCK, the MLML VAR HEADER, the MLML VAR ARRAY,and the MLML EXTENSION BLOCK.

File Header

File header information is recorded with each disk file to describe its general contents. The file header is a structure containing 8 header, 80-character, strings, which are useful to annotate the source of the data and data processing information. The use of these headers is up to the user, but complete annotation is strongly recommended. The file header also holds the file creation date. The VMS system time stamps each disk *ARCHIVE* file as part of the VMS file system each time the file is modified. An auxiliary REAL*4 array accompanies each file. The purpose of this array is up to the user. One example of this auxiliary array used with CTD data is to store the station date, time, latitude and longitude. Those values are not associated with the data in the individual *Variables* (conductivity, temperature and pressure), but with the complete data set.

PLOT also provides extensions to the database in both the *File Header* and for each *Variable*. These *Extension* data are used in specialized application with our oceanographic and marine optics data acquisition programs to hold such things as spectroradiometer sampling depth, water temperature, battery and power supply voltages, and the state of data processing. Each MLML instrument has its own *Extension* variables specific to that instrument. These *Extension* data are of little importance to most users, but it is useful to know that the PLOT database is extensible.

Table 5. Examples of PLOT batch files.

```
! STAT1_01.MLCOM, 07 May 92, Feinholz, FeLine Light/thesis data
! Plot processing for PROD_01_CTD_3383_COR into PROD_01_CTD_3383_STATS
! Before this run RADIO3.HPL (_RAW --> _COR)
! This gives mean,sdev,min,max,N stats from calibration corrected data
! (after this run STAT2.HPL, LN.HPL (finish _COR) )
! Element ranges from looking at data & stats...
!
!     u data         # pts       ------Result vars-----------------------------------------
!     v var  seq     seq all stor mean  s  min   max   n     mi me va sk ku sumx sumxx sumxxx
stats 1 1-10 133-139 *  2   3    11-20 * 21-30 31-40 41-50 *  *  *  *  * 51-6 0 61-70 71-80
stats 1 1-10 141-144 *  2   3    11-20 * 21-30 31-40 41-50 *  *  *  *  * 51-6 0 61-70 71-80
stats 1 1-10 146-148 *  2   3    11-20 * 21-30 31-40 41-50 *  *  *  *  * 51-6 0 61-70 71-80
stats 1 1-10 150-152 *  2   3    11-20 * 21-30 31-40 41-50 *  *  *  *  * 51-6 0 61-70 71-80
```

```
! RADIO3.MLCOM, 24 April 92, Feinholz, Feline Light thesis data
! Plot processing for PROD_%%_CTD_%%%%_RAW into PROD_%%_CTD_%%%%_COR
! This applies calibration corrections for MLML Radiometer #3
! (after this run STAT1_%%.HPL, STAT2.HPL, LN.HPL (_COR --> _STATS) )
!
! Lu450
transform x*60*1357 3 3 *
! Lu500
transform x*60*831.1 4 4 *
! Lu680
transform x*60*573.8 5 5 *
! Temperature
transform 0.111+1.0195*(x*60) 6 6 *
! Pressure
transform x*60 7 7 *
transform -9.238+0.00940*x+166.610*y+0.38299*x*y 6 7 7 0 *
! EsPAR
transform x*60*225.6 8 8 *
! Ed450
transform x*60*7767 9 9 *
! Ed500
transform x*60*3604 10 10 *
```

```
! MOCE1.MLCOM STN7_PRC_DERIVED seq#'s to replace w/ MISSING_REAL (MF 9Jul93)
!
! clip beginning of blue and end of red for MOS Ed,Lu,SNR,Ke,Kl,Lw's
noaa edit 3,4,7,8,11,12,15,16,19,20,23,24,27-36 1-89,662-1000
!
! clip blue/red overlap for MOS Ed,Lu,SNR,Ke,Kl,Lw's
noaa edit 3,4,7,8,11,12,15,16,19,20,23,24,27-36 454-521
```

Variable

A *PLOT Variable* is a structure containing a 1-dimensional numeric data array (sometimes called a vector) and a variety of *Variable Header* information describing these data. Most of the header information controls the appearance of graphs.

Variable Header

Each 1-dimensional numeric array is accompanied by header information describing the data. The *Variable Header* structure contains a *Description* string describing the data, a separate *Caption* string is used to annotate axes on graphs, a string containing the FORTRAN format specifier used to print data listings, 5 parameters to define graph scales (scale minima and maxima, tick interval, label frequency and format) and 6 optional parameters providing further control over the appearance of graphs. Figure 1 shows the use of each scaling parameter. All values in a single *Variable* contain the same data type. A REAL*4 fixed-length array is used to carry auxiliary data along with each *Variable*. This array is used in specialized applications.

The MLML_VAR_HEADER shown in Appendix 2 contains three elements not described above. These control the type (DATA_TYPE) and size (SIZE) of numerical data in the *Variable* and also the possibility of another REAL*4 array (AUX). DATA_TYPE can be any one of those shown in Appendix 2 (REAL*4, REAL*8, INTEGER*2, etc). The *Data Type* specifies the kind of data stored in the *Variable*. The default type is REAL*4. All values in a single *Variable* contain the same data type, but arithmetic can be performed between *Variables* containing different data types.

Most users will work with files containing *Variables* of a single data type. SIZE contains the total number of bytes in a *Variable*: it is updated whenever data are added or removed from the *Variable*. ELEMENT_SIZE is the number of bytes used by each numerical value. AUX is another extension array used to characterize each *Variable* and is of no use for general applications. Very often a *PLOT Variable* stores a data series in which each element is related to its neighbors by a sampling interval. These serial type of data may need other values (such as time of day, sampling depth, instrument voltage levels) to characterize them. Much of that can be written to the DESCRIPTION in ASCII characters, but often it is more appropriate to put those values in a numerical array such as AUX.

Data Array

The term Data Array is used generically to refer to the numeric data stored in *PLOT* files. Each data file contains a single 2-dimensional Data Array which consists of a number of 1-dimensional *Variables* (or vectors), and a number of *Elements* (or rows). Each Data Array contains data of a single type. For example, a Data Array sufficient to hold salinity, temperature and pressure data from 250 sampling depths, is dimensioned to have 3 *Variables* containing 1000 bytes each, when values are stored as REAL*4 numbers requiring 4 bytes per value.

The Data Array is contained in the structure MLML_VAR_ARRAY which contains how many *Variables* have been allocated, DIM_NUM_VAR, and their length in bytes, DIM_SIZE. Contained in this structure are the numeric data, X, and their associated *Variable Header*, V. X and V are pointers to the array, a fact which requires some understanding when writing *PLOT* programs.

Data Element

A data element is a single value stored in the Data Array, and each data element is referenced by specifying a *Variable* number (synonymous with 'column') and *Element* number (synonymous with 'row'). The default data type is REAL*4, which requires 4 bytes of disk or computer memory and contains about 7 significant digits in the range of about ± 1E37 to 1E-37. Other data types such as BYTE (8 bits), INTEGER*2 (16 bits), INTEGER*4 (32 bits), and REAL*8 (64 bits) are allowed in different variables, even within the same file. The choice of the data type to use depends on the application: the larger the size of the Data Element, the greater the precision of the data element. The *ENTER* program provides the means to change the data type of a *Variable.* Remember, also, that the data type can be set when plot is first started.

Archive File

The permanent repository of a data file is in the user's default directory. This is referred to as the *Archive* file to denote its permanence. The *PLOT* program will never change the contents of an *Archive* file, because when the user of a *PLOT* application program records results, the VMS system will create a new file, or a new version of an existing file. Only the user can purge or delete *Archive* files.

Workspace File

A *Workspace* file is used as a means to move data between different *PLOT* application programs. This temporary file is created in the user's login directory, defined by the VMS logical SYS$SCRATCH, when the user first starts a *PLOT* session. It is deleted when *PLOT* is exited gracefully. If *PLOT* is terminated abnormally by a system crash or use of CONTROL Y, *Workspace* is retained, and before beginning the next *PLOT* session, that *Workspace* including the original data can be retrieved. The *Workspace* must be large enough to contain the longest *Variable* required and should allow for several more *Variables* than required to allow buffering temporary results.

The reason *Workspace* files are needed may be understood by an example. Suppose you want to process some data (with a hypothetical *PLOT* program called MLML_PROCESS), list it, and record the revised file. The following steps are required:

1. $ PLOT 20 100 (the Workspace file is created)

2. ML> LOAD (run program to load selected Variables from Archive file)

3. ML> PROCESS (run the processing program)

4. ML> LIST (run program to list data)

5. ML> RECORD (run program to record results in a new Archive file)

6. ML> EXIT (quit the program)

DATA FOUND IN WORKSPACE
Do you want to Exit MLML_DBASE Anyway?(Y/N)

In this example, four separate programs are invoked. Data are transparently buffered in a *Workspace* file between invocation of each program. This is necessary because each program must allocate its own *Memory* space in which data are processed. The *Memory* space is allocated by each program to process as many data *Elements* as are consistent with the *Workspace* file. The idea is that *Memory* space (i.e. RAM or random access memory) must be conserved in a multiuser system, but *Workspace* (i.e. disk memory) can be made as large as a user requires.

When exiting the program using the EXIT or CTRL Z at **home base**, the user will be reminded that data remain in the *Workspace*, which can be recorded before leaving. If a PLOT session is unintentionally interrupted, the *Workspace* will be retained in the directory defined by SYS$SCRATCH. When the next *PLOT* session is started, the user will be asked whether or not to use the existing *Workspace Variables*.

Acknowledgements

Work on the MLML_DBASE programs has continued for several years through funding provided by grants from the National Oceanic and Atmospheric Administration, National Earth Satellite Data and Information Service. Recent funding has been provided under NOAA/NESDIS Grant No. NA17EC0428-1. Mike Feinholz supplied many of the examples used here, and we appreciate Marilyn Yuen's careful editing.

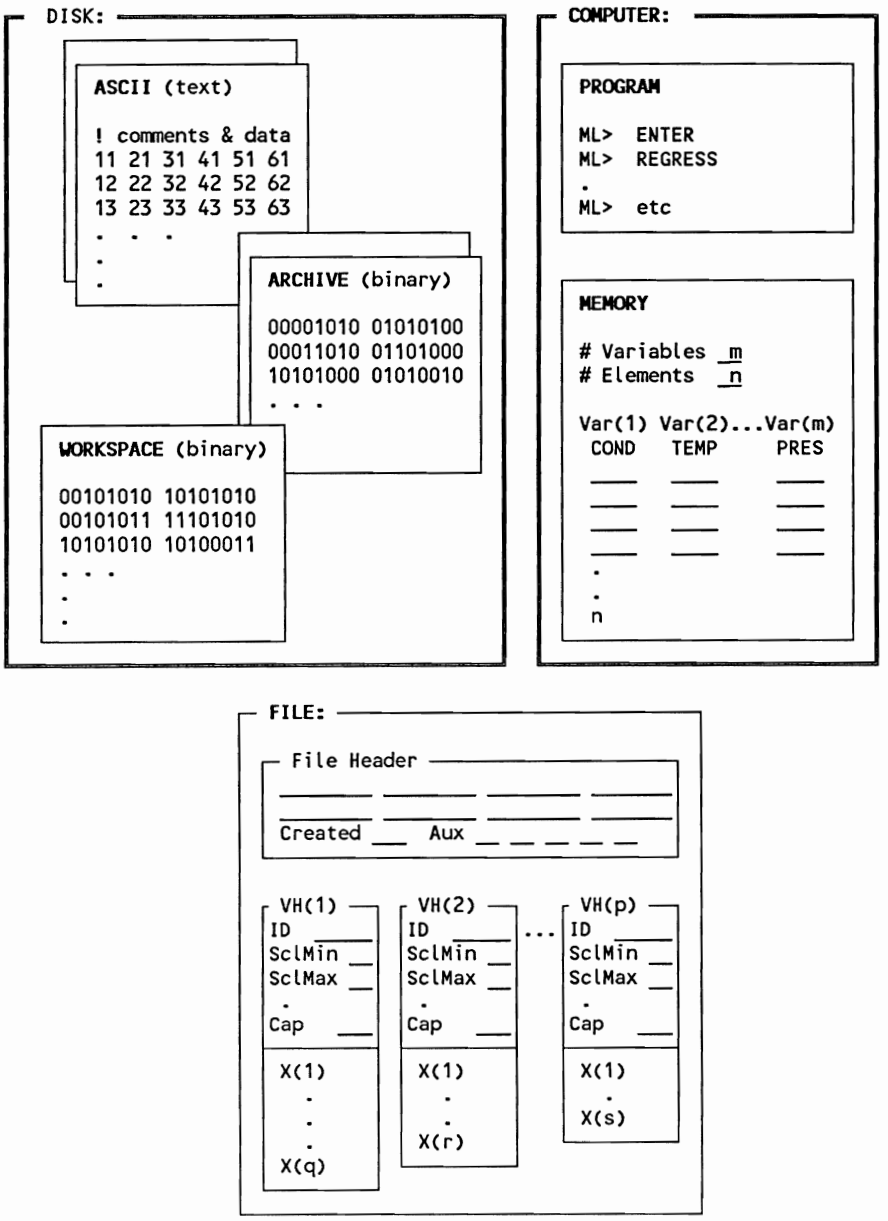


Figure 2. The MLML_DBASE file scheme. Three types of disk files may be used to hold MLML_DBASE data. ASCII are editable, human readable files. ARCHIVE and WORKSPACE are binary files that share the same structure shown in the lower diagram. The files contain *File Header* information, and any number of *Variables:* (1..p). which hold both *Variable Header* information VH(n) and the numerical data X(j). Each variable may be vectors of varying length: (1..q, 1..r, 1..s), and may be of different data types (e.g. REAL*4 or REAL*8)

Appendix 1. MLML_DBASE program menus.

PROGRAM	Main Menu Operations	
1 ENTER		
	1 = ENTER	Data
	2 = CHANGE	Data
	3 = INSERT	Data
	4 = DELETE	Data
	5 = CLEAR	Variables
	6 = COPY	Data, Captions, Scale Parameters
	7 = MOVE	Variables (Delete Old Vars)
	8 = ENTER or EDIT	Variable Description, Symbol & Format
	9 = "	Caption & Scale Parameters
	10 = "	Auxiliary Parameters
	11 = "	Variable Data Type
	12 = "	File Header
2 LIST		
	1 = LIST	Data
	2 = "	Graphics Parameters and Captions
	3 = "	Symbols, Format Parameters and Data Type
	4 = "	Auxiliary Arrays
	5 = "	File Header and Variable Description
	6 = CHANGE	Output Device or File
	7 = "	Listing Setup Parameters
	8 = "	Listing Format
3 FILE		
	1 = LOAD	Complete Variables
	2 = "	File Header only
	3 = "	Variable Headers only
	4 = "	Data only
	5 = CHAIN	Data
	6 = RECORD	Complete Variables
	7 = "	File Header only
	8 = "	Variable Headers only
	9 = "	Data Values only
	10 = LIST	File Size
	11 = "	File Extensions
	12 = "	Workspace Extensions
4 LOAD	LOAD	Complete Variables
5 RECORD	RECORD	Complete Variables
6 SIZE	RETURN WORKSPACE or ARCHIVE File Size	

Appendix 1. MLML_DBASE program menus (continued).

PROGRAM	Main Menu Operations

7 GRAPH: X-Y Line and Scatter Plot

1 = CLEAR	Display or Eject Page
2 = LABEL	and Outline Graph
3 = PLOT	Data
4 = LABEL	Titles
5 = ENTER	Scale Parameters
6 = AUTOSCALE	Scale Parameters
7 = SELECT	Graphics Parameters
8 = SELECT	Graphics Device and Port (Current = TT:)

GRAPH: 7 Produces the Following Submenu:

Enter Graphics Parameters:

1 = Plot Type and Sizes
2 = X Axis
3 = Y Axis
4 = Z Axis
5 = Font and Outline
6 = Viewport
7 = File/Date/Time Stamps
8 = Display Current Settings
9 = Reset to Default Settings

Graphics Parameters 6 Produces the Submenu

Available Viewports:

# Description	Xmin	Xmax	Ymin	Ymax
1 = 1-PANEL	0.160	0.960	0.160	0.960
2 = 2-PANEL (Horizontal) - Top	0.160	0.960	0.570	0.980
3 = " " - Bottom	0.160	0.960	0.080	0.490
4 = 2-PANEL (Vertical) - Left	0.100	0.490	0.160	0.960
5 = " " - Right	0.590	0.980	0.160	0.960
6 = 3-PANEL (Horizontal) - Top	0.045	0.985	0.720	0.992
7 = " " - Middle	0.045	0.985	0.387	0.658
8 = " " - Bottom	0.045	0.985	0.053	0.325
9 = 3-PANEL (Vertical) - Left	0.090	0.320	0.160	0.960
10 = " " - Center	0.423	0.653	0.160	0.960
11 = " " - Right	0.757	0.987	0.160	0.960
12 = 4-PANEL Lower - Left	0.060	0.490	0.080	0.490
13 = " Lower - Right	0.555	0.985	0.080	0.490
14 = " Upper - Left	0.060	0.490	0.575	0.985
15 = " Upper - Right	0.555	0.985	0.575	0.985
16 = ENTER Viewport Dimensions	0.350	0.750	0.200	0.900

Appendix 1. MLML_DBASE program menus (continued).

PROGRAM	Main Menu Operations

7 GRAPH: X-Y Line and Scatter Plot

GRAPH: 7 (SELECT Graphics Parameters) with

Graphics Parameter 8 Produces the Following Display

* * * * * CURRENT GRAPHICS PARAMETERS * * * * *

		Caption		Label	
Axis	Position	Size	Rotation	Size	Rotation
X	Bottom	6.00	0	4.50	0
Y	Left	6.00	90	4.50	0
Z	Right	4.00	90	3.00	0

Soft viewport: Xmin = 0.200 Ymin = 0.200
Xmax = 0.800 Ymax = 0.800
Viewport Rotation = 0 deg
DECWindow size = 0.60
Clip Viewport = ON
Tick Length: Major = 3.00 Minor = 1.50% of Y axis
Outline Line Width = 3.00
Grid Line Width = 2.00

Plot Border = NO
Font Type = -101 (Device dependent -- check manual)

Plot Type = '+' (Plot Symbol)
Plot Type = 'z' (Label Z Values with Markers)
LINE: Type = 1 (Solid)
Width = 3.00
Pen action = LIFTS UP on Missing Data
MARKER: Type = -1 (Solid Circle)
Size = 3.00 mm
Z LABELS: Position = Left
Size = 2.50 mm
Decimals = 0
Rotation = 0 deg
Marker Type = -1 (Solid Circle)
Marker Size = 3.00 mm
N LABELS: Size = 2.50 mm
SYMBOL: Size = 2.50 mm

Stamp	Type	X Pos	Y Pos	Size (mm)	Text
Date/Time	NONE	0.800	0.100	4.00	MLML
File	FILE NAME ONLY	0.100	0.100	4.00	

Appendix 1. MLML_DBASE program menus (continued).

PROGRAM	Main Menu Operations

GRAPH: 8 (SELECT Graphics Device) produces two Submenus

Available Graphics Devices:

#	Description	Symbol Name
1	= Falco Graphics Terminal	FALCO
2	= Tektronics 4010/4014 Graphic Terminal	TEK4014
3	= VT-340 Color Graphics Terminal	VT340
4	= HP-7475 or LVP16 (8.5"x11") HPGL Plotter	HP7475
5	= LVP16 (11"x17") HPGL Plotter	LVP16B
6	= HP-7550 HPGL Plotter	HP7550
7	= HP LaserJet @ 75 dots/inch	HPLJ_75
8	= HP LaserJet @ 100 dots/inch	HPLJ_100
9	= HP LaserJet @ 150 dots/inch	HPLJ_150
10	= HP LaserJet @ 300 dots/inch	HPLJ_300
11	= Postscript	POSTSCRIPT
12	= DECwindow Display (to remote node)	DECW
13	= GKS Metafile (GKSM)	GKSM_OUT
14	= Other Graphics Device Type	

Available Graphics Ports:

#	Description	Port Name
1	= User's Terminal	TT:
2	= HP LaserJet Printer (Oceanography Lab)	$LASER:
3	= HP LaserJet Printer (Library)	$LIBPRT:
4	= NEC Silentwriter Laser Printer	$NEC:
5	= HPGL Hardcopy Plotter (Computer Room)	$PLOTTER1:
6	= HPGL Hardcopy Plotter (Oceanography Lab)	$PLOTTER2:
7	= System Printer	$PRINTER:
8	= DECwindows Display (to remote node)	DECW$DISPLAY:
9	= File Name or other Device Name	

Appendix 1. MLML_DBASE program menus (continued).

PROGRAM	Main Menu Operations

8 TRANSFORM

Perform algebraic and oceanographic functions using MLHPL interpreter

9 MAP

1 = SELECT Map Projection and Parameters
2 = CLEAR Display or Eject Page
3 = LABEL and Outline Map
4 = PLOT World Database
5 = PLOT Grids
6 = PLOT User Data on Map
7 = LABEL Title
8 = ENTER Scale Parameters
9 = AUTOSCALE Scale Parameters
10 = SELECT Graphics Parameters
11 = SELECT Graphics Device and Port

10 CONVERT

1 = ASCII (Text) to ARCHIVE (Binary)
2 = ASCII (Text) to WORKSPACE (Binary)
3 = ARCHIVE (Binary) to ASCII (Text)
4 = WORKSPACE (Binary) to ASCII (Text)

11 REGRESS

1 = POLYNOMIAL Sum (Bi*X^i) i = 1..10
2 = MULTIPLE Sum (Bi*Xi) i = 1..10
3 = HARMONIC Sum (Ai*cos(Ni*X) + Bi*sin(Ni*X)) i = 1..5

4 = CHANGE Output Device or File
5 = PLOT Results on Existing Graph

12 STATS

1 = UNIVARIATE Statistics down columns, by Element
2 = " " down columns, by Range of Values
3 = " " across rows, by Variable
4 = ONE-WAY FREQUENCY TABLE: Analysis
5 = " " " Percentiles
6 = TWO-WAY " " Analysis
7 = GROUPED DATA Statistics
8 = F-TEST and t-TEST

9 = CHANGE Output Device or File (Current = SYS$OUTPUT)

Appendix 1. MLML_DBASE program menus (continued).

PROGRAM	Main Menu Operations	
13 TIDE		
	1 = START	Transfer Constituents
	2 = PRINT	Daily Highs and Lows
	3 = SAVE	Daily Highs and Lows
	4 = "	Predictions at even intervals
	5 = ENTER	Constituents
	6 = PRINT	Constituents
	7 = "	Station Index
	8 = CHANGE	Output Device or File (Current = SYS$OUTPUT)
14 HEADER		
	1 = PRINT	File Headers
	2 = PRINT	File Headers and Variable Descriptions
	3 = EDIT	File Headers
	4 = CHANGE	Output Device or File (Current = SYS$OUTPUT)
15 NUMER		
	1 = DIFFERENTIATE	
	2 = INTEGRATE	
	3 = INTERPOLATE	
	4 = SMOOTH	
	5 = SPECTRAL ANALYSIS	
	6 = FIND POLYNOMIAL ROOTS	
16 TOOLS		
	1 = ALIGN	Groups of Variables by Key Values
	2 = COMPACT	Remove Range of Values
	3 = DECIMATE	Retain Data in Element List
	4 = DELETE	Remove Data in Element List
	5 = INSERT	Constant between Elements
	6 = INVERT	Variables top to bottom
	7 = PAD	Data with Constant
	8 = REPLACE	Data with Constant
	9 = SORT	Variables in Ascending or Descending Order
	10 = TRANSPOSE	Rows & Columns or Elements & Variables
	11 = TRIM	Data at Beginning or End of Variable

Appendix 2. MLML_DBASE Data Structure.

```
! MLML_DBASE.FOR
! This module defines the MLML 'PLOT' database structure and parameters.
! It must be included in all FORTRAN programs using the PLOT database.

!   AUTHOR: Richard Reaves
!   DATE:   October 1988
!       rev by Reaves, 25 Sep 89, removed MAJ_TICK and added AXIS_TYPE's
!       rev by Reaves, 06 Nov 91, added additional file extension types
!       rev by Reaves, 07 Nov 91, added mlml_extension_block
!       rev by Feinholz, 28 May 92, added MOS and SIS extension types
!       rev by Reaves, 17 Dec 92, added more graph label types

! Define sizes of structure elements

    INTEGER*4   MAX_VAR_AUX, MAX_HDR_STR, MAX_HDR_AUX, MAX_EXT_DATA

    PARAMETER  (MAX_VAR_AUX  = 10,
    2       MAX_HDR_STR  = 8,
    2       MAX_HDR_AUX  = 10,
    2       MAX_EXT_DATA = 100)

! Define data types (data_type)

    INTEGER*4   MLML_DT_UNKNOWN, MLML_DT_BYTE,
    2           MLML_DT_INT2, MLML_DT_INT4,
    2           MLML_DT_REAL, MLML_DT_DBLE, MLML_DT_REAL16,
    2           MLML_DT_CREAL, MLML_DT_CDBLE, MLML_DT_CHAR

    PARAMETER   (MLML_DT_UNKNOWN = 0,    ! Can be structures, etc.
    2           MLML_DT_BYTE     = 1,    ! Integer numbers
    2           MLML_DT_INT2     = 2,
    2           MLML_DT_INT4     = 3,
    2           MLML_DT_REAL     = 4,    ! Floating point numbers
    2           MLML_DT_DBLE     = 5,
    2           MLML_DT_REAL16   = 6,
    2           MLML_DT_CREAL    = 7,    ! Complex numbers
    2           MLML_DT_CDBLE    = 8,
    2           MLML_DT_CHAR     = 9)    ! Character data

! Define missing datum also called NaN in MLHPL

    BYTE        MISSING_BYTE
    INTEGER*2   MISSING_INT2
    INTEGER*4   MISSING_INT4
    REAL*4      MISSING_REAL
    REAL*8      MISSING_DBLE
    REAL*16     MISSING_REAL16

    PARAMETER (MISSING_BYTE = '80'X,
    2       MISSING_INT2    = '8000'X,
    2       MISSING_INT4    = '80000000'X,
    2       MISSING_REAL    = 1.701100E+38,              ! Derived from IMSL
    2       MISSING_DBLE    = 1.70109999255703617E+38,   ! Derived from IMSL
    2       MISSING_REAL16  = 5.809099663209518212725903950897 4086E+4931)

! Define axis types (axis_type)

    INTEGER*4   MLML_AXIS_LINEAR, MLML_AXIS_LOG_10, MLML_AXIS_LOG_E

    PARAMETER     (MLML_AXIS_LINEAR = 0,
    2              MLML_AXIS_LOG_10 = 1,
    2              MLML_AXIS_LOG_E  = 2)

! Define label types (lbl_type)

    INTEGER*4   MLML_LBL_FIXED, MLML_LBL_FLOATING_DEC,
    2           MLML_LBL_FLOAT, MLML_LBL_10_POWER,
    2           MLML_LBL_E_POWER, MLML_LBL_MONTHS,
    2           MLML_LBL_DEGREES, MLML_LBL_DEGMIN, MLML_LBL_DEGMINSEC,
    2           MLML_LBL_HOURS, MLML_LBL_HRMIN, MLML_LBL_HRMINSEC
```

Appendix 2. MLML_DBASE Data Structure (continued).

```
    PARAMETER        (MLML_LBL_FIXED        =  0,
    2           MLML_LBL_FLOATING_DEC =  1,
    2           MLML_LBL_FLOAT        =  2,
    2           MLML_LBL_10_POWER     =  3,
    2           MLML_LBL_E_POWER      =  4,
    2           MLML_LBL_MONTHS       =  5,
    2           MLML_LBL_DEGREES      = 10,
    2           MLML_LBL_DEGMIN       = 11,
    2           MLML_LBL_DEGMINSEC    = 12,
    2           MLML_LBL_HOURS        = 20,
    2           MLML_LBL_HRMIN        = 21,
    2           MLML_LBL_HRMINSEC     = 22)

! File Information Data Types

    INTEGER*4   MLML_FILE_FIX, MLML_FILE_VAR

    PARAMETER        (MLML_FILE_FIX  = 0,    ! Fixed variable length
    2           MLML_FILE_VAR  = 1)    ! Variable length for each variable

! File and Variable Extension Data Types

    INTEGER*4   MLML_EXT_END, MLML_EXT_AUX, MLML_EXT_GRAPH,
    2           MLML_EXT_STR

    PARAMETER   (MLML_EXT_END   = 0,     ! No more extensions available
    2           MLML_EXT_AUX    = 1,     ! Auxiliary array
    2           MLML_EXT_GRAPH  = 2,     ! Graphic extensions
    2           MLML_EXT_STR    = 3)     ! An additional string

! Additional File and Variable Extension Types

    INTEGER*4   MLML_EXT_CTD_HEADER,    MLML_EXT_CTD_INFO,
    2       MLML_EXT_MOS_HEADER,    MLML_EXT_MOS_INFO,
    2       MLML_EXT_SIS_HEADER,    MLML_EXT_SIS_INFO,
    2       MLML_EXT_FASTIE_HEADER, MLML_EXT_FASTIE_INFO

    PARAMETER  (MLML_EXT_CTD_HEADER    = 100,   ! Ctd header (stn pos, time, etc.)
    2       MLML_EXT_CTD_INFO      = 101,   ! Ctd info for each variable (calibs, data type, etc.)
    2       MLML_EXT_MOS_HEADER    = 200,   ! MOS Spect header (stn pos, time, etc.)
    2       MLML_EXT_MOS_INFO      = 201,   ! MOS Spect info for each variable (calibs, data type, etc.)
    2       MLML_EXT_SIS_HEADER    = 300,   ! SIS Spect header (stn pos, time, etc.)
    2       MLML_EXT_SIS_INFO      = 301,   ! SIS Spect info for each variable (calibs, data type, etc.)
    2       MLML_EXT_FASTIE_HEADER = 400,   ! FASTIE radiometer header (stn pos, time, etc.)
    2       MLML_EXT_FASTIE_INFO   = 401)  ! FASTIE radiometer info for each variable (calibs, data type,etc.)
```

Appendix 2. MLML_DBASE Data Structure (continued).

```
! MLML Dbase structures

    STRUCTURE    /MLML_VAR_HEADER/
         CHARACTER*80   DESCRIPTION     ! string explaining the variable contents or processing status
        CHARACTER*8     SYMBOL          ! symbol used in data listings; for symbolic arithmetic in TRANSFORM
        CHARACTER*20    FMT   ! FORTRAN syntax for format listing parameter
        INTEGER*4       FMT_WIDTH       ! column width from FMT
        CHARACTER*32    CAPTION         ! caption to annotate graph axis

! Basic Graph Parameters
        REAL*4          MIN_SCL         ! axis minimum scaling value
        REAL*4          MAX_SCL         ! axis maximum scaling value
        REAL*4          TICK_INT        ! tick interval in axis scale units
        INTEGER*4       LBL_INT         ! number of ticks between numerical labels
        INTEGER*4       DEC             ! # decimal places to write labels

! Graphic Extensions
        INTEGER*4       AXIS_TYPE       ! type of axis
        INTEGER*4       LBL_TYPE        ! type of labels
        INTEGER*4       LBL_MIN         ! label count from scale min
        INTEGER*4       LBL_MAX         ! label count from scale max
        INTEGER*4       GRID_TICK       ! number of tick marks between grids
        INTEGER*4       GRID_DENS       ! grid density (dots/tick)

        REAL*4          AUX(MAX_VAR_AUX)  ! auxiliary array use is not defined in MLML_DBASE

        INTEGER*4       SIZE            ! the current variable length in bytes
        INTEGER*4       DATA_TYPE       ! data type contained in variable
        INTEGER*4       ELEMENT_SIZE    ! the number of bytes per element
    END STRUCTURE

    STRUCTURE  /MLML_VAR_ARRAY/
        INTEGER*4       DIM_NUM_VAR     ! the maximum number of variables in the workspace
        INTEGER*4       DIM_SIZE        ! the maximum number of bytes in the workspace
        INTEGER*4       X               ! data elements are stored here
        INTEGER*4       V               ! variable header
    END STRUCTURE
                                ! Note the elements X and V are pointers

    STRUCTURE   /FILE_HEADER_BLOCK/
        CHARACTER*80    HSTR(MAX_HDR_STR)  ! file identifiers strings like H$ in HPL PLOT
        CHARACTER*20    CREATED            ! file creation dated
        REAL*4          AUX(MAX_HDR_AUX)   ! auxiliary file array
    END STRUCTURE

! The File or Variable Extension defined below is a completely flexible data structure that can be used for
! presently unimagined purposes.  From the previous page note the four File Extension data Types
! are defined for specific data acquisition needs.

    STRUCTURE   /MLML_EXTENSION_BLOCK/
        INTEGER*4       TYPE               ! type of data in file extension
        INTEGER*4       SIZE               ! dimensioned size of file extension array in bytes
        INTEGER*4       DATA               ! pointer to the file extension data
    END STRUCTURE
```

Appendix 2. MLML_DBASE Data Structure (continued).

```
! Definitions for the file and subprocess interfaces

    CHARACTER*(*)    DEFAULT_WORKSPACE_FILENAME,
    2           DEFAULT_WORKSPACE_LOGICAL,
    2           DEFAULT_FILENAME_LOGICAL,
    2           DEFAULT_PRINTER_LOGICAL,
    2           DEFAULT_PARAMETER_LOGICAL,
    2           GRAPHIC_PARAMETER_LOGICAL,
    2           GRAPHIC_PARAMETER_FILENAME,
    2           DEFAULT_LOGICAL_TABLE,
    2           DEFAULT_FILENAME_TYPE

    PARAMETER  (DEFAULT_WORKSPACE_FILENAME = 'SYS$SCRATCH:MLML_DBASE_WORKSPACE',
    2       DEFAULT_WORKSPACE_LOGICAL   = 'MLML_DBASE_WORKSPACE_FILE',
    2       DEFAULT_FILENAME_LOGICAL    = 'MLML_DBASE_FILE_NAME',
    2       DEFAULT_PRINTER_LOGICAL     = 'MLML_DBASE_PRINTER',
    2       DEFAULT_PARAMETER_LOGICAL   = 'MLML_DBASE_PARAMETER',
    2       GRAPHIC_PARAMETER_LOGICAL   = 'MLML_GRAPH_PARAM_FILE',
    2       GRAPHIC_PARAMETER_FILENAME  = 'SYS$LOGIN:MLML_GRAPH_PARAMETERS.DAT',
    2       DEFAULT_LOGICAL_TABLE       = 'LNM$JOB',
    2       DEFAULT_FILENAME_TYPE       = '.MLDAT')


! Union structure for extracting or saving data

    STRUCTURE /MLML_EQUIV_DATA/
        UNION
            MAP
                BYTE        B(16)
            END MAP
            MAP
                INTEGER*2   W(8)
            END MAP
            MAP
                INTEGER*4   I(4)
            END MAP
            MAP
                REAL*4      R(4)
            END MAP
            MAP
                REAL*8      D(2)
            END MAP
            MAP
                REAL*16     Q
            END MAP
            MAP
                COMPLEX*8   CR(2)
            END MAP
            MAP
                COMPLEX*16  CD
            END MAP
            MAP
                CHARACTER   C*16
            END MAP
        END UNION
    END STRUCTURE
```

Appendix 3. Index of MLML Library Functions and Subroutines.

The subprograms listed here are available on MLML VAX computers and workstations in LNK$LIBRARY. Many of them are written specifically for the MLML_DBASE programs. Others provide common oceanographic functions: all of the functions listed in SCOR/UNESCO Publications 42 and 44 are included. Superscripts for the routines indicate the following: 1 = functions applicable to C programs only; 2 = Function returns a single precision (REAL*4) result and companion function with D prefix returns a double precision (REAL*8) result; 3 = function produces only double precision result. Refer to the MLML Subroutine Library for specifics on application and examples.

Data Entry Routines

The following routines provide error checking so that erroneous entries may be flagged by the calling program.

ENTER_CHAR
Prompts user for character input including non-printable characters such as ^C or ^Z

ENTER_DBLE
Prompts user for double precision (REAL*8) input

ENTER_DBLE_HPL
Prompts user for expression to be interpreted by MLHPL, which returns a double precision (REAL*8) value

ENTER_INT
Prompts user for integer (REAL*4) input

ENTER_INT_HPL
Prompts user for expression to be interpreted by MLHPL, which returns an integer (INTEGER*4) value

ENTER_REAL
Prompts user for REAL*4 input

ENTER_REAL_HPL
Prompts user for expression to be interpreted by MLHPL, which returns a single precision (REAL*4) value

ENTER_STR
Prompt user for string input

YES_OR_NO
Prompts user for logical or boolean input

Math Functions and Subroutines

APPLY_PRESSURE_RESPONSE
Computes ideal wave pressure transfer function from sample interval, bottom depth, pressure sensor depth in REAL*8 vector

DEG_RAD [2]
Converts angular units from degrees to radians

DETREND
Detrend a REAL*8 data series by least squares (for spectral analysis)

DROUND [2]
Returns input rounded to the given decimal place

FFT
Calls FFT_WORK and IFT_WORK routines, returns the frequency domain series from time domain input

FFT_NPAIR
Determines maximum number of pairs for a fast Fourier transform from the number of values

FFT_POWER
Computes the power spectra estimates and phase from the complex Fourier coefficients

FFT_WORK
Performs the direct Fourier transform

FRAC360
Returns the fractional part of a real number as degrees, eg. FRAC360(3.25) = 90

GC_DIST[2]
Returns the great circle distance; inputs in decimal degrees

GC_INTERPOLATE[2]
Returns interpolated position given distance

GEOGR_RECT [2]
Conversion of magnitude and geographic direction (in degrees) to north and east vector components

GREAT_CIRCLE [2]
Computes the great circle distance and heading; Inputs may be either DD.mmm or decimal degrees

HEAP_SORT
Sort REAL*8 array by Heapsort method

HEAP_SORT_INDEX
Sort INTEGER*4 array indexing the REAL*8 array by Heapsort

IFT
Calls FFT_WORK and IFT_WORK and returns the time domain series from frequency domain input

IFT_WORK
Performs the inverse fast Fourier transform

POLAR_RECT [2]
Conversion of magnitude and direction (in radians) to vector components

POLAR_RECTD [2]
Conversion of magnitude and direction (in degrees) to vector components

PRESSURE_RESPONSE
Returns the pressure response function for Ideal Wave period and transducer depth

PROUND [2]
Returns input rounded to the given base 10 power

QUICK_SORT
Sort REAL*8 array by Quicksort method

RAD_DEG [2]
Converts angular units from radians to degrees

RECT_GEOGR [2]
Conversion of north and east vector components to magnitude and geographic direction (in degrees)

RECT_POLAR [2]
Convert vector components to magnitude and direction (in radians)

RECT_POLARD [2]
Convert vector components to magnitude and direction (in degrees)

SHELL_SORT
Sort REAL*8 array by Shell's method

SMOOTH_SPECTRAL_ESTIMATES
Apply the Hamming smoothing for spectral estimates in REAL*8 vector

STRAIGHT_SORT
Sort REAL*8 array by straight insertion

String Functions and Subroutines FORTRAN

CREATE_PROMPT
Fills string to end with periods. Used to align prompts.

SLEN
Returns number of characters in a string

STRCAP
Convert string to uppercase

STRFLOAT [2]
Converts a floating point number to a string with trailing zeros truncated

STRLWC
Convert string to lowercase

STRREPL
Replace characters in substring

STRREV
Reverse characters in substring

ZAP_SPACES
Remove space characters in string

String Functions and Subroutines C

STRFLT [12]
Converts a floating point number to a string with trailing zeros truncated

STRLOWER[1]
Converts string to lowercase characters

STRPOS[1]
Returns position of substring in a string

STRLTRIM [1]
Removes leading whitespaces

STRRTRIM [1]
Removes trailing whitespaces

STRTRIM [1]
Removes leading and trailing whitespaces

STRUPPER [1]
Converts string to uppercase characters

STRZAP [1]
Removes all whitespaces from string

Oceanographic Functions

All algorithms are taken from Unesco Reports. For historical perspective, the simple Knudsen-Ekman equation of state and Wilson's velocity of sound equations are included.

Fofonoff, N.P. and R.C. Millard Jr. (1983) Algorithms for computation of fundamental properties of seawater. Unesco Technical Papers in Marine Science 44.

Postma, H., A. Svansson, A. Lacombe, and K. Grasshoff (1976)International Oceanological Tables for Oxygen Solubility in Sea Water. Oceanology 15:240-241

Carbon dioxide sub-group of the joint panel on oceanographic tables and standards. (1983) Unesco Technical Papers in Marine Science 42.

ATG_83 [2]
Adiabatic temperature gradient

BO4K_83 [2]
Computes K1 dissociation constant for boric acid

BULK_MODULUS_83 [2]
Secant bulk modulus of seawater from S,T,P

CO2K1_83 [2]
Returns K1 dissociation constant for carbon dioxide

CO2K2_83 [2]
Computes K2 dissociation constant for carbon dioxide

CO2SOL_83 [2]
Carbon dioxide solubility from salinity and temperature

COND_78 [2]
Electrical conductivity from salinity, temperature, pressure

DELTA_83 [2]
Specific volume anomaly from salinity, temperature, pressure

DENSITY_83 [2]
Density from salinity, temperature, pressure

DEPTH_83 [2]
Depth as function of pressure and latitude

EKMAN_MU [2]
Seawater compressibility f(S,T,P) Ekman's 1908 equations

FREEZE_83 [2]
Freezing point of seawater

KNUDSEN [2]
Sigma-t f(S,T) Knudsen's (1901) equations

KNUDSEN_EKMAN [2]
In-situ seawater density, Knudsen-Ekman (1901-1908) equations

LAB_SAL [2]
Salinity from conductivity ratio (Rt) and temperature

OXYSOL_76 [2]
Oxygen solubility from salinity and temperature

SALIN_78 [2]
Salinity from conductivity ratio (R), temperature, pressure

SIGMA_83 [2]
Sigma notation density from salinity, temperature, pressure

SPC_HT_83 [2]
Specific heat at constant pressure

SVEL_83 [2]
Velocity of sound in seawater

THETA_83 [2]
Potential temperature of seawater

WILSON_SOUND [2]
Wilson's simple velocity of sound in seawater f(S,T,P)

Ephemeris and Time/Date Routines

AIR_MASS [2]
Computes the air mass for given altitude uses Kastenen's iterative method for low altitudes

ALT_AZ [2]
Computes the altitude and azimuth of celestial objects from position and time

CHECK_HMS
Checks for invalid minute/second combinations

CHECK_YMD
Checks for invalid month/day combinations

CLEAR_SKY [2]
Computes the clear sky PAR or total sea level irradiance

DATE_YMD [2]
Extracts integer Year, Month and Day from the DD.MMYY format

DCALEN_DATE [3]
Converts Julian date to Calendar date (double precision only)

DEPHEM_PARM [3]
Provides input arguments for SUN_ALMANAC subroutine (double precision only)

DJULIAN_DATE [3]
Converts Calendar date to Julian date (double precision only)

LAN [2]
Computes Local Apparent Noon of the Sun

RISE_SET [2]
Computes the GMT time of sunrise or sunset

SUN_ALMANAC [2]
Computes ephemeris parameters for given date and time

TIME_HMS [2]
Extracts integer Hours, Minutes, Seconds from the HH.MMSS format

TO_DEG [2]
Converts angle (or time) from DD.MMSS to decimal degrees

TO_DMS [2]
Converts decimal degrees to DD.MMSS or DD.MMm

Mapping Procedures

The following procedures use algorithms taken from Snyder, J.P. 1982. Map Projections used by the U.S. Geological Survey. Geological Survey Bulletin No. 1532. The satellite view algorithm was adapted from GrafKit. World map databases were adapted from GrafKit (approximately 5 nautical mile resolution) and C.I.A. database (approximately 1 km resolution) from University of Miami DSP software.

MAP_PROJECTION
Initializes internal variables according to map projection types for use with MAP_CONVERT and MAP_INVERT. Arguments used by this routine depend upon the type of projection: Equidistant Cylindrical, Mercator, Miller Cylindrical, Mollweide, Azimuthal Equidistant, Gnomonic, Lambert Azimuthal Equal Area, Orthographic, Stereographic, Satellite View, Albers Equal Area Conic, Lambert

MAP_OPEN
Opens up a graphic device and prepare parameters for plotting. The viewport will be adjusted according to map projection and the window coordinates given.

MAP_CLOSE
Empties the graphic output buffer and closes the graphic device

MAP_PLOT
Plots latitude and longitude to the graphic device. Note these plotting coordinates are saved in a buffer and the program must execute MAP_PLOT_END to empty the buffer.

MAP_PLOT_END
Terminate plotting latitude-longitude pairs by emptying buffer used in MAP_PLOT

MAP_GRIDDING
Plot latitude-longitude grids from latitude grid spacing, longitude grid spacing, reference latitude and longitude where grid begins, bit pattern for grid line, length of line pattern in degrees

MAP_WORLD_DATA
Plot world database adapted from according to border types such as lands (i.e. continent outline, islands, etc.), waters (i.e. lakes, rivers, etc.), and political boundaries. Resolution in nautical miles may be specified to reduce plotting time.

MAP_CONVERT
Converts map positions (latitude and longitude) to rectangular X and Y coordinates for plotting.

MAP_INVERT
Converts from rectangular X and Y coordinates to latitude and longitude. This routine is useful for digitizing.

System Routines

CPUTIME [3]
Returns the process accumulated CPU time

DELETE_LOGICAL
Deletes a system logical

FPARSE [1]
Parse a VMS file specifier into Node, Device, Directory, FileName, FileType and Version; C programs

FSEARCH [1]
Check default or other VMS directory for presence or absence of target file

FVALID [1]
Checks filenames for valid VMS filename characters

GET_LOGICAL
Translate a logical name to VMS logical table?

GETUIDNAME [1]
Returns the User Identification for the current process

MEMAVAIL
Returns the available number of bytes of memory

PARSE_FILESPEC
Used in FORTRAN programs to parse a VMS file specifier into Node, Device, Directory, FileName, FileType and Version

RTIME [3]
Returns real time in seconds since 0 January 1970

SAVE_LOGICAL
Creates or modifies a logical

SEARCH_FILE
Used in FORTRAN programs to check default or other VMS directory for presence or absence of target file

I/O Procedures

These procedures control the flow of data between the computer and peripheral devices such as printers, modems and terminals.

ABORT_TRANSFER
Stops and cleans up transfer to a device

ALLOCATE_DEVICE
Allocates a device to the process

BUFFER_BUSY
Returns logical TRUE, if transfer is in progress

BUFFER_DATA
Returns length of data in buffer

BUFFER_RESET
Resets the pointer in the buffer and clears buffer

BUFFER_SPACE
Returns available buffer space

DEALLOCATE_DEVICE
Deallocates a device or all devices

DEVICE_ALLOCATED
Returns TRUE if device is allocated

IOBUFFER
Establish buffer in memory

KEY
Return a keycode from a keyboard interrupt

KEY_TABLE
Converts a string of escape sequences to a VT-220 keycode

OFF_EOT
Disables interrupt upon end of transfer

OFF_KEY
Disables the keyboard interrupt

ON_EOT
Enables interrupt upon end of transfer

ON_KEY
Enables the keyboard interrupt

READBUFFER
Reads a character from a buffer

READCHAR
Reads a byte value from terminal port or VMS device

READSTRING
Reads a string from terminal port or VMS device

SET_TIMEOUT
Sets the timeout period for read or write port

TRANSFER
Sets up buffer to transfer to/from a device

TRANSFER_UNTIL
Sets up buffer to transfer to or from a device until a terminating character is received

WRITEBUFFER
Writes a character to a buffer

WRITECHAR
Writes a character to a terminal port or VMS device

WRITESTRING
Writes a string of characters to a terminal port or VMS device

Serial I/O Subroutines

CLEAR_LINES
Clears the terminal port hardware lines

HANGUP_MODEM
Hangs up a terminal port

SENSE_LINES
Returns the status of the terminal port hardware lines

SENSE_SERIAL
Returns the terminal port serial port parameters such as baud rate, parity, and data bits

SENSE_TERMINAL_CHAR
Returns permanent characteristics of a terminal port such as echo and wrap flags

SENSE_TERMINAL_MODE
Returns current terminal characteristics such as columns, lines/page

SET_LINES
Sets the terminal port hardware lines

SET_SERIAL
Sets the terminal port hardware parameters such as baud rate, data bits, and parity

SET_TERMINAL_MODE
Set terminal characteristics such as echo and wrap flags

SET_TERMINAL_SIZE
Set terminal width and page length

TERMINAL_SIZE
Returns the page length and width of the terminal port

MLML_DBASE Procedures

The following procedures are not of much use in stand alone programs, since their purpose is to manipulate MLML_DBASE variables as shown in Appendix 1. Before these routines can be used effectively the MLML_DBASE data structure must be understood.

Setting up MLML_DBASE Variables

CLOSE_WORKSPACE
Closes the WORKSPACE file to prevent further writing

DELETE_WORKSPACE_EXTENSION
Removes a specific header extensions from WORKSPACE

DIM_MLML_DATA
Dimensions the data array in computer MEMORY

FREE_MLML_DATA
Frees up the MEMORY allocated by DIM_MLML_DATA

OPEN_WORKSPACE
Opens the WORKSPACE file to speed up disk access

REMOVE_MLML_WORKSPACE
Deletes the WORKSPACE file and logicals

RESET_WORKSPACE_VAR
Resets and clears the WORKSPACE variables to the initial state

SETUP_MLML_WORKSPACE
Allocates the WORKSPACE file and creates logicals

WORKSPACE_VAR_SIZE
Returns the size, data type and element size of a variable in the WORKSPACE file

MLML_DBASE Archive File Operations

CLOSE_MLML_FILE
Closes a disk ARCHIVE file to prevent further writing

GET_MLML_FILENAME
Returns the complete file specification of a ARCHIVE file

MLML_FILE_SIZE
Returns the number of Variables and maximum number of data values in an ARCHIVE file

MLML_VAR_SIZE
Returns the size, data type and element size of an ARCHIVE variable

OPEN_MLML_FILE
Opens or creates a disk ARCHIVE file

Move data between MLML Workspace and Memory

ALLOCATE_AND_LOAD_DATA
Copies data from WORKSPACE into MEMORY based on the Variables contained in a number list

CLEAR_WORKSPACE_EXTENSION
Set the workspace extensions to zero

QUERY_WORKSPACE_EXTENSION
Gets the extension information from the file header or variable.

READ_WORKSPACE_EXTENSION
Reads header extensions from WORKSPACE to MEMORY

READ_FILE_HEADER
Copies the file header with extensions from WORKSPACE to MEMORY

READ_VAR
Copies a variable header with or without the data from WORKSPACE to MEMORY

MLML_WORKSPACE_SIZE
Returns the dimensioned size of the WORKSPACE file

WRITE_FILE_HEADER
Copies a File header with extensions from MEMORY to WORKSPACE

WRITE_VAR
Copies a Variable header with or without the data from MEMORY to WORKSPACE

WRITE_WORKSPACE_EXTENSION
Copies header extensions from MEMORY to WORKSPACE

Moving data between Archive Files and Memory

LOAD_FILE_HEADER
Copies a file header with extensions from an ARCHIVE file to MEMORY

LOAD_VAR_DATA
Copies Variable Data and Variable Header from ARCHIVE to MEMORY

LOAD_VAR_HEADER
Copies a Variable Header with extensions from ARCHIVE to MEMORY

QUERY_EXTENSION
Obtain extension information from an ARCHIVE Variable or File Header

RECORD_FILE_HEADER
Copies only File Header with extensions from MEMORY to ARCHIVE

RECORD_VAR_DATA
Copies Variable Data and Variable Header from MEMORY to ARCHIVE

RECORD_VAR_HEADER
Copies only Variable Header with extensions from MEMORY to ARCHIVE

Manipulate MLML_DBASE values in MEMORY

GET_DATA
Copies data from a DBASE Variable to a vector or array in an application program

GET_DATUM
Copies a single data value from a DBASE Variable to an application program variable

GET_DATUM_STR
Converts a numeric data value to a string based on that Variable's format specification

SAVE_DATA
Copies values from an application program vector or array to a DBASE Variable

SAVE_DATUM
Copies a single data value from an application program variable to a DBASE Variable

SHIFT_DATA
Shifts data within a DBASE Variable by a shift count and fills any resulting gaps with a constant

SWAP_DATA
Exchanges two data values in a DBASE Variable

Manipulate MLML_DBASE Variables

AUTOSCALE
Setup scale parameters according to data in Variables

ENTER_CAPTION_AND_SCALES
Prompts user for input of DBASE Variable caption and plotting scales

ENTER_FORMAT_PARAMETER
Prompts user for DBASE format parameter input (used to list MLML_DBASE values), and checks its validity

ENTER_ELIST
Prompts user for string input to be interpreted as a element list, and checks its validity

ENTER_ENUM
Prompts user for an element number and checks its validity

ENTER_VAR_DESCRIPTION
Prompts user for DBASE Variable description input

ENTER_VAR_SYMBOL
Prompts user for DBASE Variable symbol input

ENTER_VLIST
Prompts user for Variable List input and checks its validity

ENTER_VNUM
Prompts user for DBASE Variable Number input, checks for valid range

GET_LAST_ELEMENT
Returns the last Element Number in a Variable

MLML_CONVERT_DATA_TYPE
Converts data in a Variable from one type to another

MLML_ELEMENT_SIZE
Returns element size in bytes depending on data type

PARSE_FMT_PARAM
Parses and saves the listing format spec to Variable Header, checks for validity

SORT_DBASE
Sort MLML_DBASE variables by Heapsort

Performing MLML_DBASE Graphics Operations

Graphics are drawn using the DEC Graphics Kernel System (GKS). These subroutines listed here package the low level GKS subroutines in a graphics environment more similar to that used in HPL PLOT.

BORDER
Draws a border around the plotter frame

CONVERT_TO_VIEWPORT
Converts X and Y values to viewport coordinates before plotting

DISPLAY_GRAPH_PARAM
Displays the current graph parameter settings

DRAW_AXES
Plots both the X and Y axes with major and minor ticks including grids based on DBASE scale parameters

DRAW_AXIS
Plots the either the X or Y axis with major and minor ticks based on the DBASE scale parameters

DRAW_GRIDS
Plots grids on the X and Y axes based on DBASE scale parameters

ENTER_GRAPH_PARAM
Prompts user for graph scaling parameters input (plot and line type, rotation of axes, caption position, viewport position, etc)

GET_AXIS_LABEL
Returns string containing the month, mantissa, or exponent based upon the current label type

GET_CHAR_SIZE
Returns the height and width of a character in world units

GET_GRAPHIC_SETUP
Reads the "GRAPHICS_DEVICE.DAT" file to verify availability of the input graphics device

GET_PLOTTER_INFO
Reads hardware setup parameters (eg. P1 and P2) from HPGL plotters

GKS_CLOSE
Close the GKS buffer and for POSTSCRIPT printer sends file to designated device

GKS_FLUSH
Outputs whatever remains in the GKS buffer and in case of output to terminal pause until RETURN

GKS_OPEN
Opens a GKS graphics port and sets the workstation viewport to the maximum device coordinates

LABEL_AXIS
Labels the X or Y axis based on the current values of the graphics scale parameters. This routine defines the appearance of an MLML_DBASE graph.

LOAD_GRAPH_PARAM
Load the graphics parameters from a binary file named in the MLML_GRAPH_PARAM logical

SAVE_GRAPH_PARAM
Saves the current graph scaling parameters to the file named in the logical MLML_GRAPH_PARAM_FILE

SELECT_GRAPHIC_DEVICE
Selects a graphics device from the list in GRAPHIC_DEVICES.DAT

SELECT_GRAPHIC_PORT
Selects a graphic output port from the list in GRAPHIC_PORTS.DAT

SELECT_GRAPHIC_VIEWPORT
Selects the graphics viewport from the list in GRAPHIC_VIEWPORTS.DAT or a file named in the logical MLML_GRAPHIC_VIEWPORT _FILE, or allows manual entry of the graphic viewport

SELECT_PRINTER
Selects a printer from the list available printers

SET_CHAR_SIZE
Set the size of characters in mm or % of plotting area (as appropriate) and character rotation degrees

SET_DEFAULT_GRAPH_PARAM
Resets the graphics parameters (plot and line type, rotation of axes, caption position, viewport position, etc) to their default settings

SET_MARKER_SIZE
Set the GKS marker to size in mm or % of plotting area

SET_SCALES
Sets the GKS window or viewport to the DBASE graphicsparametersXMIN,XMAX,YMIN,YMAX

SET_VIEWPORT
Sets the GKS viewport on the selected device in normalized coordinates

VALID_COLOR
Checks to see if the selected pen number is within the list of valid choices

Perform MLML_DBASE Parameter File Functions

These parameters allow MLML_DBASE programs to be run from the command line or from a batch file. They consist of an ASCII string stating the program name or number, the menu item and responses to program prompts.

GET_PARAM_BOOLEAN
Reads a LOGICAL*4 parameter from the DBASE parameter file

GET_PARAM_CHAR
Reads a BYTE*1 parameter from the DBASE parameter file

GET_PARAM_DBLE
Reads a REAL*8 parameter from the DBASE parameter file

GET_PARAM_INT
Reads an INTEGER*4 from the DBASE parameter file

GET_PARAM_REAL
Reads a REAL*4 parameter from the DBASE parameter file

GET_PARAM_STR
Reads a CHARACTER*(*) parameter from the DBASE parameter file

OPEN_PARAM_FILE
Opens a parameter file for reading parameters

Numberlist Routines

The numberlist routines parse an ASCII string to form a series of integers that designate the range of DBASE Variable Numbers or Data Elements. The filelist routines are similar, and they return a filename based on a sequence number, or range of sequence numbers.

GET_FILELIST
Called in FORTRAN programs to return a file name as a character string from user input of a file sequence number. Use of this procedure reduces the number of keystrokes to manipulate files. The *Alias* user file names are contained in a file named filename.ALIAS.

GET_FLIST [1]
Called in C programs to return a file name as a character string... see above

GET_NUMLIST
Extracts a value from a Number List

MAX_NUMLIST
Returns the maximum value in a Number List

MIN_NUMLIST
Returns the minimum value in a Number List

PARSE_NUMLIST
Parses string and return numbers from list

SAVE_NUMLIST
Packs an integer array into a Number List

SORT_NUMLIST
Sorts entries in a Number List in ascending or descending order

TRUNCATE_NUMLIST
Retains values in the number list based on the input range and removes outliers

MLHPL Routines

The MLHPL routines implement the HPL interpreter, which allows use of command line algebra in MLML_DBASE.

COMPILE_HPL
Converts an HPL line or program into tokens

DECOMPILE_HPL
Converts HPL tokens into text

EXECUTE_HPL
Executes HPL compiled tokens

FULL_REAL
Converts an 8-byte HPL full precision BCD value to a REAL*8

HPL_EMSG
Returns an error message from the error numbers

HPL_XREF
Returns a cross reference of HPL variables

REAL_FULL
Converts a REAL*8 value binary value to an HPL full precision 8-byte BCD value

REAL_SPLIT
Converts a REAL*8 value binary value to an HPL split precision 4-byte BCD value

SPLIT_REAL
Converts a 4-byte BCD HPL split precision value to a REAL*8 binary value